UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA "RENATO M. CAPOCELLI"

CORSO DI DOTTORATO IN INFORMATICA
XI CICLO – NUOVA SERIE

ANNO ACCADEMICO 2011-2012

TESI DI DOTTORATO IN INFORMATICA

Secure Computation Under Network and Physical Attacks

Tutor
prof. **Ivan Visconti**

Candidata
Alessandra Scafuro

Coordinatore
prof. **Giuseppe Persiano**

Contents

Introduction

The aim of cryptography is to provide methods and techniques to prove security of a protocol in a rigorous manner. Typically, security concerns the ability of preserving the secrecy of the inputs of the parties running the protocol. The first, crucial, step towards proving security of a protocol is to have a formal definition of security.

Although everyone has an intuitive notion of what security should mean, it turns out to be very difficult to formally define it. For example, consider the basic task of encryption. Our intuitive notion of security for an encryption scheme is that, a scheme is good if any eavesdropper, when given a ciphertext, is not able to "understand" the plaintext. The first difficulty here is to define what we mean by saying that an adversary "cannot understand". One way to define this, is to say that the adversary cannot *recover* the plaintext. Namely, given a ciphertext, the adversary cannot compute the plaintext that is hidden. However, this notion seems to be not very robust. Indeed, a protocol in which the eavesdropper is able to extract few bits of the plaintext (but is not able to recover the whole plaintext), would be still secure according to such definition. Instead, what we really wanted is that an adversary can not extract *any* information from the ciphertext. How to formally capture such intuitive notion?

The way we define security of an encryption scheme is through the concept of *indistinguishability* [50]. Very roughly, we say that an encryption scheme is secure if the encryption of 0 is "indistinguishable" from the encryption of 1.

In general, the task of providing formal definitions that capture the security requirements that we desire to achieve in real-life protocol is an highly non-trivial task. Thus when moving from the basic task of encryption to the general task of secure computation, providing a robust security definition that accurately captures the real-world scenario, is even more tricky.

The setting of secure computation is the following. There are many entities, that we call parties. Parties want to jointly compute a function of their inputs, while keeping their input secret. As an example of this setting, consider a voting system. Each party has as secret input a preference, i.e., the name of the person that the party wants to vote. The joint function run by the parties takes as input all the secret preferences (the votes) and computes the majority. As it is already apparent from this example, besides the correctness of the result, an important property that we require from a voting system is that the preference of each player remains private. Therefore, the intuitive definition for

secure computations would be the following. A protocol securely computes a function if the output of the protocol is the correct evaluation of the function on the inputs of the parties, and the privacy of the input of each party is preserved. This is still not quite precise. Indeed, for some functionalities, the output of the function is such that it reveals the inputs played by the party. (As an example, consider the function SUM computed by two players only, clearly, from the knowledge of the output, and one private input, each player can compute the secret input of the other party). Therefore, a more precise definition of security requires that, a protocol is secure if each player does not learn anything more than what is leaked from the knowledge of the output and the knowledge of its own private input.

Note that this setting is quite different from the set of encryption seen before where there are two parties, trusting each other, and they want to preserve the privacy of their communication against a third party that does not participate[1] to the protocol, but it only observes the messages sent over the channel. In the setting of secure computation all parties are *mutually distrusting*. Therefore, each party might try to actively deviate from the protocol in order to learn more information.

How can we formally model the concept of not learning anything in this setting? A brilliant idea introduced in [51] is the concept of the simulator. The idea is that anything that an adversary can learn by interacting with the actual parties participating in the protocol, it can be learned also by the simulator, that is not interacting with any real party but it simulates the protocol executions in his head using fake inputs (for the honest real parties). Informally, to prove that a protocol is secure, is sufficient to show that, for any adversarial party (or set of parties) attacking the protocol in a real world execution, interacting with the honest players, there exists a simulator that successfully performs the very same attack, but executing the protocol in his head and without knowing the input of the honest players.

The above definition of secure computation is a good starting point, but it is still very far from capturing real-life setting. Indeed, in the setting above we have considered a very restricted adversary, that participates to one protocol execution only, and that can misbehave by corrupting a sub-set of parties. Thus, proving security in this setting, implies that a protocol will be secure as long as it will be run in isolation, namely parties cannot be involved in the execution of other protocols.

However, in real-life scenarios many functionalities are run over the internet, therefore a party participating in one protocol execution is by default connected and involved in many other protocol executions, simultaneously. Thus, in the real world an adversary is not actually running one execution of the protocol in isolation, but it can activate and interleave many executions of other (or the same) protocols. Hence, in the formal security definition one should model the adversary as an entity that initiates and actively participates in many protocol executions, that can be run concurrently.

Such general security definition was introduced in [16] and is called Universal Composability. In the Universal Composability framework, besides the simulator and the

[1] In more demanding definitions of encryption a party might play as man in the middle to try to infer some information.

adversary seen before, a new entity called "environment" is introduced. While the simulator and the adversary still model one execution of the target protocol for which one wants to prove security, the purpose of the environment is to model the concurrent executions of other (possibly different) protocols, that are run together with the target protocol. The environment can of course communicate with the adversary. This models the fact that while attacking the target protocol, the adversary can exploit the information gained from the concurrent executions. The Universal Composability (UC) framework is very general, and thus provides very strong security guarantees. Therefore, is desirable to have protocols that can be proved secure in this model. Unfortunately, it has been shown in [21] that it is impossible to design a protocol that achieves such definition, without the help of some setup assumption. A setup can be seen as some tool that parties can use to run the protocol. A setup is *trusted* if it is assumed that the adversary cannot participate in the generation of such setup. For example, a trusted setup can be that each party is given a smart card, and the adversary cannot participate in the process of generating/delivering such smart cards. Therefore, the behavior of such smart card is never adversarial.

Given that the UC definition cannot be achieved without assuming some setup, two lines of research have been explored. One line of research investigated on the possibility of relaxing the need of trust in the setup assumptions required to achieve UC-security. For example, in the setup that we discussed above, the assumption was that each party receives a trusted smart-card. However, assuming that all the smart cards, even the ones used by the adversary are trusted seems unrealistic, and it seems indeed natural to ask whether we can *reduce* the amount of trust needed to achieve UC-security. This line of research, started with [62], explores the use of hardware boxes (formally, tamper-proof hardware tokens) in the protocol. The idea is that, each party must trust only its own hardware box, and not the boxes generated/delivered by the other players.

The second line of research instead investigates on relaxing the UC-definition so that it is possible to design protocols without using any trusted setup. Such relaxed definitions capture a restricted scenario in which an adversary activates many executions of the *same* protocol simultaneously, playing always the *same* rule (in contrast in the UC-definition the adversary can execute arbitrary protocols concurrently, and play different rules). We refer to this more relaxed definition as security in the concurrent setting, and to the adversary playing in this setting, as a concurrent adversary. Besides the mere feasibility results, this line of research develops on understanding the minimal requirements, such as round and communication complexity, computation complexity, black-box uses of cryptographic primitive, for a protocol which is secure in the concurrent setting.

Our Contribution

In this thesis we provide contributions for both lines of research. In the first part of the thesis we discuss how to achieve UC-security based on physical setup assumptions while removing the trust in any trusted third party, and the trust on the physical devices used by the adversary. We explore the use of Physically Uncloneable Functions (PUFs) as setup assumption for achieving UC-secure computations. PUF are physical noisy source

of randomness. The use of PUFs in the UC-framework has been proposed already in [14]. However, this work assumes that all PUFs in the system are *trusted*. This means that, each party has to trust the PUFs generated by the other parties. In this thesis we focus on reducing the trust involved in the use of such PUFs and we introduce the Malicious PUFs model in which only PUFs generated by honest parties are assumed to be trusted. Thus the security of each party relies on its own PUF only and holds regardless of the goodness of the PUFs generated/used by the adversary. We are able to show that, under this more realistic assumption, one can achieve UC-secure computation, under computational assumptions. Moreover, we show how to achieve *unconditional* UC-secure commitments with (malicious) PUFs and with stateless tamper-proof hardware tokens. We discuss our contribution on this matter in Part I. These results are contained in papers [80] and [28].

In the second part of the thesis we focus on the concurrent setting, and we investigate on protocols achieving *round optimality* and *black-box* access to a cryptographic primitive. We study two fundamental functionalities: commitment scheme and zero knowledge, and we study two concurrent attack models, as explained below.

Commitment scheme is a two-stages (commitment, decommitment stage) functionality run between a committer and a receiver. The committer has a secret value in input, and it commits to such value running the commitment stage. When the committer is ready to reveal the value to the receiver, it runs the decommitment stage, also called opening stage. The security properties of a commitment scheme are hiding and binding. Hiding preserves the security of the committer, and is the property of the commitment stage. A commitment stage is hiding if the secrecy of the input committed is preserved against any adversarial receiver. Binding is a correctness property of the commitment scheme, and it requires that for a given transcript of the commitment stage, there exists only one value that can be revealed by any, possible adversarial, committer. In this thesis we will consider the following attack model for commitment schemes. An adversarial receiver can interleave arbitrarily many commitment sessions, and once the commitment stages are over, ask for the decommitment (the opening) of some of them, adaptively on the transcript observed from the sessions played so far and in any order. Note that this adversary is not purely concurrent, since it cannot ask to open a session if the commitment stage of all the sessions is completed. Namely, there is a barrier between the commitment phase and the openings. We refer to this type of concurrency as concurrency with barrier. This attack model is referred in literature as Selective Opening Attack (SOA, in short) and was introduced in [39]. A commitment scheme secure in this model is said to be SOA-secure.

A Zero Knowledge protocol is run between two parties, a prover and a verifier. Both parties have as common input an instance x of an **NP** language L. The prover has as input a witness w for the statement $x \in L$, and he wants to use knowledge of the witness to convince the verifier that the statement is true, however, he does not want to reveal any information about the witness. Security here means that any (concurrent) adversarial verifier running a protocol execution with the prover on input x should not gain any information besides the fact that $x \in L$. The correctness requirement establishes that

if the statement is false, i.e., $x \notin L$, any adversarial (possibly concurrent) prover should not be able to convince a honest verifier of the truthfulness of the statement. In this thesis we consider the following attack model. An adversarial verifier can initiate any polynomial number of protocol sessions in concurrency, but it is bounded on the number of identities that it can play with. More precisely, we consider a setting in which any verifier that wishes to run the zero knowledge protocol with the prover, has to register its identity in a public file before any proof begins. Thus, there is a registration phase, in which no interaction between prover and verifier takes place, but each verifier upload its public identity in a common public file. After the registration has been completed, the "proof phase" starts. In this phase prover and registered verifiers run concurrently many proofs. The restriction is that the proof phase can be run only by registered verifiers. This model is called Bare Public Key (BPK, for short) model, and was introduced in [19].

In this thesis we consider both models, and we focus on some of the round-optimal constructions and lower bounds concerning both functionalities and both models proposed in literature. Our findings is that such constructions present subtle issues. Hence, we provide new protocols that actually achieve the security guarantees promised by previous results.

More specifically, concerning SOA-secure commitment schemes, we provide a construction that requires only 3 rounds for the commitment stage (the decommitment state is non interactive) and that makes black-box use of any two-round trapdoor commitment scheme. The round complexity of our construction, contradicts the previous lower bound shown in [100], and is round optimal for SOA-secure commitment schemes. Thus, we demonstrate that the lower bound for SOA-secure commitment scheme is 2 round instead of 3. We also show that fully concurrent [2] SOA-secure commitment scheme do not exist, regardless of the black-box use of cryptographic primitives, but under the assumption that security is proved via a black-box (thus rewinding) simulator. This result contradicts the security of the fully concurrent SOA-secure scheme shown in [100]. This contribution is shown in the paper [79].

Concerning zero knowledge protocols in the BPK model, we show that all previous *round-optimal* constructions for concurrent zero knowledge in the BPK model, suffer of a subtle issue. Specifically, we show that there exists an adversarial verifier's strategy, that makes any (black-box) simulator fail. We show that in many cases this issue cannot be fixed by trivial changes of the protocol, or by providing a new simulation strategy. Therefore, we propose a new construction that is round optimal and for which we can show a simulation strategy that is successful against any adversarial verifier. Such results are discussed in Part II, and are the content of the paper [95].

Physical Attacks

So far we have discussed about network attacks. Namely, we have considered an adversary which exploits the network to run concurrent executions of different protocols.

[2] Fully concurrent means that the adversarial receiver is allowed to interleave commitment and decommitment phase of different sessions.

We now turn to physical attacks. Namely, we consider an adversary that can physically tamper with the machine of the honest party while running the protocol. In particular, in this thesis we consider a very specific attack in which the adversary is able to *reset* the memory of the machine of an honest party, in fact forcing the machine to run many executions of the same protocol *reusing* the same randomness. At first sight, this attack might seem too unrealistic, as one can imagine that the honest party can physically protect its own machine from the adversary. Therefore, considering such attack when designing a protocol, might seem an overkill. However, as nowadays tiny and weak computational devices are used in cryptographic protocols, such attack has become a real threat. Smart cards are the canonical example of tiny devices computing (sensitive) cryptographic protocol. Indeed, we use smart card everyday to perform bank transactions or for identification purposes. The way computations with smart card work, is that we put our smart card in some more powerful computing device, like a smart-card reader, or a PC, and they run a protocol. In this scenario resetting attacks seem very plausible, since a smart card is a very weak device compared to a reader or to a computer.

The formal security definition capturing such attack has been introduced in [19]. This definition focuses only on the zero knowledge functionality (and the weaker notion of witness indistinguishability), and considers only the case in which only one party is reset. More precisely, the party which is subject to reset is always the prover. Thus, the constructions shown in [19] are secure if the prover is subject to reset attack, but is not secure if the verifier is reset instead. Later in [7] resettable security base been formulated for the opposite case. Namely, [7] provides constructions that are secure only if the party that is subject to reset is the verifier. Clearly, an interesting question is whether one can design a protocol in which the party that might be subject to reset attack is not known in advance. Therefore, such protocol should be secure in both cases. Such notion is called security under *simultaneous resettability*. The work of [29] provides a protocol that is secure in such setting. Namely, it presents a construction for simultaneously resettable zero knowledge system. However this result does not close the gap between what we are able to achieve in presence of a fixed resetting party, and what we can achieve when both parties can reset. Indeed, in case of one-side resetting (either only the prover can reset, or only the verifier can reset), we know how to achieve protocols that are *arguments of knowledge*. Roughly, a protocol is an argument of knowledge, if the prover can convince the verifier of a truthfulness of a statement, if and only if it knows the witness.

In this thesis we provide the first construction of a witness indistinguishable argument system that is *simultaneous resettable* and *argument of knowledge*. We discuss about this contribution in Part III, which is the content of the paper [24].

Roadmap. Chapter 1 is devoted to the set up of the notation and the definitions of the cryptographic primitives used in this thesis.

In Part I we present our results for achieving UC-security using (malicious) PUFs. Specifically, in Chapter 2 we present our malicious PUF model, and our construction for achieving general secure two-party computation using malicious PUFs and computational assumptions. Then, in Chapter 3 we show how to achieve *unconditional* security

using malicious PUFs, and tamper-proof stateless tokens.

In Part II we discuss about round-optimal protocols achieving security in the concurrent setting. Specifically, in Chapter 4 we focus on round-optimal commitment schemes secure in presence of selective opening attacks. We discuss the issues in previous constructions and lower bounds, and we propose new schemes that establish new lower bounds. In Chapter 5 we instead consider round-optimal concurrently secure zero-knowledge protocols in the BPK model that have been proposed in literature. We first discuss the subtle issue in the security proof of all such protocols, and then we propose a new scheme that is round-optimal, concurrent zero-knowledge and concurrent sound and does not suffer of such issue.

Finally, in Part III we answer the challenging question of achieving argument of *knowledge* in the simultaneous resettable setting by showing the first constant-round witness indistinguishable argument of knowledge construction in this setting.

- Chapter 1 -

Preliminaries

This chapter is devoted to the set up of the notation and the definitions of the cryptographic primitives used in this thesis.

In the last section of the chapter we recall the Universally Composable Framework introduced by [16].

Notation. We denote by $n \in \mathbb{N}$ the security parameter, and by ϵ a negligible function. A function ϵ is *negligible* iff for all constants c there exists n_0 such that for all $n > n_0$ it holds that $0 \leq \epsilon(n) < 1/n^c$. We denote by PPT the property of a probabilistic algorithm whose number of steps is polynomial in its security parameter. We denote by $x \leftarrow \mathcal{D}$ the sampling of an element x from the distribution $\mathcal{D}$ and by $x \leftarrow D(z)$ the output of the algorithm D on input z assigned to the variable x. We also use $x \xleftarrow{\$} \mathsf{D}$ to indicate that the element x is uniformly sampled from set D.

We denote by $\mathsf{view}_A(A(a), B(b))(x)$ the view of A of the interaction with player B, i.e., its values is the transcript $(\gamma_1, \gamma_2, ..., \gamma_t; r)$, where the γ_i's are all the messages exchanged and r is A's coin tosses. Let P_1 and P_2 be two parties running protocol (A,B) as sub-routine. When we say that party "P_1 runs $\langle A(\cdot), B(\cdot)\rangle(\cdot)$ with P_2" we always mean that P_1 executes the procedure of party A and P_2 executes the procedure of party B. We use the notation $\{\beta_1, \ldots, \beta_k : \alpha\}$ to specify the probability distribution of α after the sequential executions of events $\beta_1, \ldots, \beta_k$. We denote by $a||b$ the concatenation between strings a and b.

In this thesis we use the work committer and sender interchangeably. In all our constructions we also allow adversaries to obtain auxiliary inputs although for simplicity we will omit them from our notation.

1.1 Commitment Schemes

In this section we provide the formal security definitions for commitment scheme, trapdoor commitment scheme and weak-trapdoor commitment scheme. As we shall see, such cryptographic primitives are used in almost all the constructions presented in this thesis.

In the following definitions we assume that parties are stateful and that malicious parties obtain auxiliary inputs, although for better readability we omit them.

Definition 1 (Bit Commitment Scheme)**.** *A commitment scheme is a tuple of PPT algorithms* $\mathsf{Com} = (\mathsf{Gen}, \mathsf{C}, \mathsf{R})$ *implementing the following two-phase functionality.* Gen *takes as input a random* n*-bit string* r *and outputs the public parameters* pk*. Given to* C *an input* $b \in \{0,1\}$*, in the first phase (*`commitment phase`*)* C *interacts with* R *to commit to the bit* b*; we denote this interaction as* $\langle \mathsf{C}(pk, \mathtt{com}, b), \mathsf{R}(\mathtt{recv}) \rangle$*. In the second phase (*`opening phase`*)* C *interacts with* R *to reveal the bit* b*, we denote this interaction as* $\langle \mathsf{C}(\mathtt{open}), \mathsf{R}(\mathtt{open}) \rangle$ *and* R *finally outputs a bit* b' *or* $\perp$*. Consider the following two experiments:*

Experiment $\mathbf{Exp}^{\text{binding}}_{\mathsf{Com},\mathsf{C}^*}(n)$*:*	**Experiment** $\mathbf{Exp}^{\text{hiding-}b}_{\mathsf{Com},\mathsf{R}^*}(n)$*:*
R *runs* $(pk) \leftarrow \mathsf{Gen}(r)$ *and sends* pk *to* C^**;*	$pk \leftarrow \mathsf{R}^*(1^n)$*;*
$\langle \mathsf{C}^*(pk, \mathtt{com}, b), \mathsf{R}(\mathtt{recv}) \rangle$*;*	$(\cdot, b') \xleftarrow{\$} \langle \mathsf{C}(pk, \mathtt{com}, b), \mathsf{R}^*(\mathtt{recv}) \rangle$*;*
$(\cdot, b_0) \xleftarrow{\$} \langle \mathsf{C}^*(\mathtt{open}, 0), \mathsf{R}(\mathtt{open}) \rangle$*;*	*output* b'*.*
rewind C^* *and* R *back after the second step;*	
$(\cdot, b_1) \xleftarrow{\$} \langle \mathsf{C}^*(\mathtt{open}, 1), \mathsf{R}(\mathtt{open}) \rangle$*;*	
output 1 *iff* $\perp \neq b_0 \neq b_1 \neq \perp$ *.*	

$\mathsf{Com} = (\mathsf{Gen}, \mathsf{C}, \mathsf{R})$ *is a commitment scheme if the following conditions hold:*

Completeness. *If* C *and* R *are honest, for any* C*'s input* $b \in \{0,1\}$ *the output of* R *in the opening phase is* $b' = b$*.*

Hiding. *For any PPT malicious receiver* R^**, there exists a negligible function* ϵ *such that the following holds:*

$$\mathbf{Adv}^{\text{hiding}}_{\mathsf{Com},\mathsf{R}^*} = |\mathsf{Pr}\left[(\mathbf{Exp}^{\text{hiding-0}}_{\mathsf{Com},\mathsf{R}^*}(n) \to 1) \right] - \mathsf{Pr}\left[\mathbf{Exp}^{\text{hiding-1}}_{\mathsf{Com},\mathsf{R}^*}(n) \to 1) \right]| \leq \epsilon(n)$$

Binding. *For any PPT malicious sender* C^* *there exists a negligible function* ϵ *such that:*

$$\mathsf{Pr}\left[\mathbf{Exp}^{\text{binding}}_{\mathsf{Com},\mathsf{C}^*} \to 1 \right] \leq \epsilon(n)$$

The above probabilities are taken over the choice of the randomness r *for the algorithm* Gen *and the random coins of the parties. A commitment scheme is statistically hiding (resp., binding) if hiding (resp., binding) condition holds even against an unbounded malicious Receiver (resp., Sender).*

The above definition is a slight modification of the one provided in [11, 59] and is more general in the fact the it includes the algorithm Gen used by R to generate the parameters for the commitment. Such a definition is convenient when one aims

to use commitment schemes as sub-protocols in a black-box way. However, for better readability, when we construct or use as sub-protocol a commitment scheme that does not use public parameters we refer to it only as $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ omitting the algorithm Gen.

Remark 1 (Binding definition)**.** *The binding property states that there exists no efficient* C^* *that can produce two distinct accepting openings for the same commitment phase with non-negligible probability. Since we consider also interactive decommitments, we formalize this notion as a game following the definition given in [11, 59]. That is,* C^* *is run twice in the decommitment phase, but with an additional input necessary to obtain two distinct openings (indeed* C^* *is run twice with the same randomness), i.e.,* C^* *is invoked as* $\mathsf{C}^*(\mathtt{open}, b)$*.*

1.1.1 Trapdoor Commitment Scheme

For the definitions of trapdoor commitments we borrow some notation from [69, 90].

Definition 2 (Trapdoor Commitment)**.** *A tuple of PPT algorithms* $\mathsf{TC} = (\mathsf{TCGen}, \mathsf{C}, \mathsf{R}, \mathsf{TCFakeDec})$ *is a trapdoor commitment scheme if* TCGen*, on input a random* n*-bit string* r*, outputs a public key/secret key pair (pk,sk),* TCGen_{pk} *is the related functionality that restricts the output of* TCGen *to the public key,* $(\mathsf{TCGen}_{pk}, \mathsf{C}, \mathsf{R})$ *is a commitment scheme, and* $(\mathsf{C}, \mathsf{TCFakeDec})$ *are such that:*

Trapdoor Property. *There exists* $b^\star \in \{0,1\}$*, such that for any* $b \in \{0,1\}$*, for all* $(pk, sk) \leftarrow \mathsf{TCGen}(r)$*, and for any PPT malicious receiver* R^* *there exists a negligible function* ϵ *such that the following holds:*

$$\mathbf{Adv}^{\mathsf{trapdoor}}_{\mathsf{TC},\mathsf{R}^*} = \mathsf{Pr}\left[\,\mathbf{Exp}^{\mathsf{trap}}_{\mathsf{TC}}(n) \to 1\,\right] - \mathsf{Pr}\left[\,\mathbf{Exp}^{\mathsf{Com}}_{\mathsf{TC}}(n) \to 1\,\right] \leq \epsilon(n)$$

The probability is taken over the choice of r *for the algorithm* TCGen *and the random coins of the players.*

Experiment $\mathbf{Exp}^{\mathsf{Com}}_{\mathsf{TC}}(n)$ *:*	**Experiment** $\mathbf{Exp}^{\mathsf{trap}}_{\mathsf{TC}}(n)$*:*
R^* *chooses a bit* b*;*	R^* *chooses a bit* b*;*
$\langle \mathsf{C}(pk, \mathtt{com}, b), \mathsf{R}^*(pk, sk, b, \mathtt{recv})\rangle$*;*	$(\xi, \cdot) \leftarrow \langle \mathsf{C}(pk, \mathtt{com}, b^\star), \mathsf{R}^*(pk, sk, b, \mathtt{recv})\rangle$*;*
$(\cdot, b') \xleftarrow{\$} \langle \mathsf{C}(\mathtt{open}), \mathsf{R}^*(\mathtt{open})\rangle$*;*	$(\cdot, b') \xleftarrow{\$} \langle \mathsf{TCFakeDec}(sk, \mathtt{open}, b, \xi), \mathsf{R}^*(\mathtt{open})\rangle$*;*
output b'*;*	*output* b'*;*

In the experiment $\mathbf{Exp}^{\mathsf{trap}}_{\mathsf{TC}}(n)$ C runs the procedure of the honest sender on input $b^\star$. The variable ξ contains the randomness used by C to compute the commitment phase and it is used by $\mathsf{TCFakeDec}$ to compute the decommitment. The knowledge of the trapdoor is required only in decommitment phase. In the trapdoor commitment of Pedersen [85], the trapdoor property holds for any $b^\star$, namely one can use the honest sender procedure to commit an arbitrary bit $b^\star$ and use the trapdoor to decommit to

any $b \neq b^\star$. Instead, in the trapdoor commitment proposed by Fiat and Shamir [43], as we show next, the trapdoor property holds only if the honest procedure was used to commit to bit $b^\star = 0$. In both commitment schemes the trapdoor is used only in the decommitment phase.

We stress that, according to the standard definition, while the hiding property must hold for all pk possibly maliciously generated by R^*, the trapdoor property must hold only for the pairs (pk, sk) honestly generated. In some definition [90] it is required that hiding holds for all the malicious keys that pass the test of an additional verification algorithm TCVer, however, w.l.o.g. one can assume that the commitment procedure runs the verification algorithm as a first step. Note that implementations of a trapdoor commitment enjoying all the properties above do exist, one example is Pedersen's Trapdoor Commitment [85], in which once the public parameter pk are given, the commitment procedure is non-interactive. We mention below a construction based on any OWF.

Trapdoor Commitment Scheme based on Non-black-box use of OWF. In [43] Feige and Shamir presented a construction of trapdoor commitments based on NBB access to OWFs. The commitment procedure consists of running the simulator of Blum's protocol [12] for Graph Hamiltonicity (HC) where the challenge is the bit to commit to. For completeness we recall the construction. Let $f : \{0,1\}^n \to \{0,1\}^*$ be a OWF.

- $(G, C) \leftarrow \mathsf{TCGen}(r)$: pick a random x and compute $y \leftarrow f(x)$. From y obtain a hard instance $G \in \mathsf{HC}$ and let C be one of the Hamiltonian cycles of G. This transformation requires non-black-box access to f. Set the public key $pk = G$ and the trapdoor $sk = C$.

- $\mathsf{C}(G, \mathtt{com}, b)$: if $b = 0$, pick a random permutation π and commit to $\pi(G)$. If $b = 1$, commit to a random n-vertex cycle H. Both commitments are performed using Naor Commitment [77] that is based on BB access to OWFs.

- $\langle \mathsf{C}(\mathtt{open}, b), \mathsf{R}(\mathtt{open}) \rangle$: C sends b and the opening according to b, i.e., if $b = 0$ it sends π and opens all commitments, if $b = 1$ it opens the cycle H. R checks whether the openings are correct according to challenge b and the procedure of the verifier of Blum's protocol.

- $\xi \leftarrow \mathsf{C}(G, \mathtt{com}, b^\star)$: C runs as $\mathsf{C}(G, \mathtt{com}, 0)$. The variable ξ contains the randomness used to run C.

- $\langle \mathsf{TCFakeDec}(C, \mathtt{open}, b, \xi), \mathsf{R}(\mathtt{open}) \rangle$: to open to 0 send π and open all the commitments, to open to 1 open the cycle C in $\pi(G)$. The opening information is taken from ξ.

Hiding comes from the hiding of Naor commitments, and binding from the hardness of the OWF. A commitment can be equivocated only if it was computed following the procedure to commit 0. Thus, the above protocol satisfies the trapdoor property for $b^\star = 0$.

Weak Trapdoor Commitment Schemes

It is possible to consider a weaker[1] definition of trapdoor commitment in which, in order to be able to later equivocate, the trapdoor is needed already in the commitment phase. The trapdoor commitment proposed in [84] satisfies such a definition.

Definition 3 (Weak Trapdoor Commitment)**.** *A tuple of PPT algorithms* $\mathsf{wTCom}=(\mathsf{wTCGen}, \mathsf{C}, \mathsf{R}, \mathsf{TCFakeCom}, \mathsf{TCFakeDec})$ *is a weak trapdoor commitment scheme if* TCGen *on input a random* n*-bit string* r*, outputs a public key/secret key pair (pk,sk),* wTCGen_{pk} *is the related functionality that restricts the output of* wTCGen *to the public key,* $(\mathsf{wTCGen}_{pk}, \mathsf{C}, \mathsf{R})$ *is a commitment scheme and* $\mathsf{TCFakeCom}, \mathsf{TCFakeDec}$ *are such that:*

Weak Trapdoor Property. *For any* $b \in \{0,1\}$*, for all* $(pk, sk) \leftarrow \mathsf{TCGen}(r)$*, for any PPT malicious receiver* R^* *there exists a negligible function* ϵ *such that the following holds:*

$$\mathbf{Adv}^{\mathsf{wtrap}}_{\mathsf{wTCom},\mathsf{R}^*} = \Pr\left[\mathbf{Exp}^{\mathsf{wTrap}}_{\mathsf{wTCom}}(n) \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{Com}}_{\mathsf{wTCom}}(n) \to 1\right] \leq \epsilon(n)$$

The probability is taken over the choice of the randomness r *for the algorithm* TCGen *and the random coins of the parties.*

Experiment $\mathbf{Exp}^{\mathsf{Com}}_{\mathsf{wTCom}}(n)$*:*	*Experiment* $\mathbf{Exp}^{\mathsf{wTrap}}_{\mathsf{wTCom}}(n)$*:*
run $\langle \mathsf{C}(pk, \mathtt{com}, b), \mathsf{R}^*(pk, sk, b, \mathtt{recv})\rangle$*;*	*run* $(\xi, \cdot) \xleftarrow{\$} \langle \mathsf{TCFakeCom}(pk, sk, \mathtt{com}), \mathsf{R}^*(pk, sk, b, \mathtt{recv})\rangle$*;*
$b' \xleftarrow{\$} \langle \mathsf{C}(\mathtt{open}), \mathsf{R}^*(\mathtt{open})\rangle$*;*	$b' \xleftarrow{\$} \langle \mathsf{TCFakeDec}(\mathtt{open}, b, \xi), \mathsf{R}^*(\mathtt{open})\rangle$*;*
output b'*;*	*output* b'*;*

As before, the variable ξ denotes the state shared by algorithms $\mathsf{TCFakeCom}$ and $\mathsf{TCFakeDec}$.

It is possible to show that there exists a non-interactive weak trapdoor commitment schemes that is *not* a "regular" non-interactive trapdoor commitment scheme as follows. Take any "regular" trapdoor commitment scheme in which the decommitment phase is non-interactive. A non-interactive weak trapdoor commitment scheme can be constructed by using the regular trapdoor commitment scheme to commit to a bit, and then by adding two (perfectly binding) commitments of the openings. The honest sender will open one of the two perfectly binding commitment chosen at random. Instead knowledge of the trapdoor from the commitment phase allows one to commit both to a decommitment of 0 and to a decommitment of 1 (in random order), therefore allowing equivocation in the opening. The interesting point is that this scheme is not a "regular" trapdoor commitment scheme, which implies that a weak trapdoor commitment scheme

[1] The definition is weaker in the sense that it is implied by the previous definition, but could be a strictly weaker primitive achievable under better assumptions and with better efficiency.

could be in theory constructed under better assumptions, or with better efficiency. Notice that in [84] it is shown a construction of an interactive weak trapdoor commitment scheme (they called it "look-ahead" trapdoor commitment) from any black-box use of a one-way permutation.

1.2 Interactive Protocols

An interactive proof (resp., argument) system for a language L is a pair of probabilistic polynomial-time interactive algorithms $\mathcal{P}$ and $\mathcal{V}$, satisfying the requirements of *completeness* and *soundness*. Informally, completeness requires that for any $x \in L$, at the end of the interaction between $\mathcal{P}$ and $\mathcal{V}$, where $\mathcal{P}$ has as input a valid witness for $x \in L$, $\mathcal{V}$ rejects with negligible probability. Soundness requires that for any $x \notin L$, for any (resp., any polynomial-sized) circuit $\mathcal{P}^*$, at the end of the interaction between $\mathcal{P}^*$ and $\mathcal{V}$, $\mathcal{V}$ accepts with negligible probability.

Formally, we have the following definitions.

Definition 4 (Indistinguishability [50])**.** *Let X and Y be countable sets. Two ensembles $\{A_x\}_{x\in X}$ and $\{B_x\}_{x\in X}$ are computationally (statistically) indistinguishable over X if for every probabilistic polynomial-time (resp. unbounded) "distinguishing" machine D there exists a negligible function $\epsilon(\cdot)$ so that for every $x \in X, y \in Y$:*

$$|\mathsf{Pr}\,[\,D(x, A_x) = 1] - \mathsf{Pr}\,[\,[\,]\,D(x, B_x) = 1\,]\,| < \epsilon(|x|).$$

Definition 5 (Interactive Proof System [52])**.** *A pair of interactive algorithms $(\mathcal{P}, \mathcal{V})$ is an interactive proof system for a language L, if $\mathcal{V}$ is PPT and there exists a negligible function ϵ such that the following conditions hold:*

- *Completeness.* $\forall x \in L$, $\mathsf{Pr}\,[\,[\,]\,(\mathcal{P}, \mathcal{V})(x) = 1] > 1 - \epsilon(|x|)$.
- *Soundness.* $\forall x \notin L$, $\forall$ *interactive machine* $\mathcal{P}^*$, $\mathsf{Pr}\,[\,[\,]\,(\mathcal{P}^*, \mathcal{V})(x) = 1] < \epsilon(|x|)$.

In case the soundness condition is required to hold only against computationally bounded prover, the pair $(\mathcal{P}, \mathcal{V})$ is called an interactive argument *system and $\mathcal{P}$ gets the witness as auxiliary input.*

Zero knowledge. The classical notion of zero knowledge has been introduced in [52]. In a zero-knowledge argument system a prover can prove the validity of a statement to a verifier without releasing any additional information. This concept is formalized by requiring the existence of an expected polynomial-time algorithm, called the *simulator*, whose output is indistinguishable from the view of the verifier.

Definition 6. *An interactive argument system $\langle \mathcal{P}(\cdot,\cdot), \mathcal{V}(\cdot)\rangle$ for a language L is computational (resp., statistical, perfect)* zero-knowledge *if for all polynomial-time verifiers $\mathcal{V}^*$, there exists an expected polynomial-time algorithm S such that the following ensembles are computationally (resp., statistically, perfectly) indistinguishable:*

$$\mathsf{view}_{V^*}((\mathcal{P}(w), \mathcal{V}^*(z))(x))_{x\in L, w\in W(x), z\in\{0,1\}^*} \text{ and } \{\mathsf{S}(x, z)\}_{x\in L, z\in\{0,1\}^*}.$$

Definition 7. *(adapted from [10] with negligible knowledge error) A proof (resp., argument) system* $\langle P(\cdot), V\rangle(x)$ *for an* **NP***-language* L *is a* proof (resp., argument) system of knowledge *if there exists a probabilistic polynomial-time algorithm* E *such that for every (resp., every polynomial-sized) circuit family* $\{P_n^*\}_{n\in N}$*, there exists a negligible function* ν *such that for any* x *of size* n*, if* $\mathsf{Pr}\left[\,\mathsf{out}(\langle P_n^*, V\rangle(x)) = 1\,\right] = p(|x|)$*, then* $\mathsf{Pr}\left[\,w{\leftarrow}E^{P_n^*(x)}(x) : R_L(x,w) = 1\,\right] = |p(|x|) - \nu(|x|)|$ *and the expected running time of* E *is polynomial in* $|x|$*.*

Witness Indistinguishability. Witness indistinguishability is a weaker security notion for interactive protocols. It requires that the view of any malicious verifier is independent of the witness used by the honest prover. This notion makes sense only when there exist more witnesses for the same theorem.

Definition 8 (Witness Indistinguishable Proof of Knowledge System $\mathcal{WI}$PoK [44])**.** *A proof of knowledge system* $(\mathcal{P}, \mathcal{V}, \mathbf{E})$ *for a* **NP** *language* L *and with witness relation* $\mathcal{R}_L$*, is witness-indistinguishable if for every PPT malicious verifier* $\mathcal{V}^*$*, there exists a negligible function* ϵ *such that,* $\forall x, \forall y_0, y_1 \in \mathcal{R}_L(x)$ *and* $z \in \{0,1\}^*$*:*

$$\mathsf{Pr}\left[\,\langle \mathcal{P}(x, y_0), \mathcal{V}^*(x, z)\rangle = 1\,\right] - \mathsf{Pr}\left[\,\langle \mathcal{P}(x, y_1), \mathcal{V}^*(x, z)\rangle = 1\,\right] < \epsilon(|x|)$$

The probability is taken over the coin tossed by $\mathcal{V}^*$ *and* $\mathcal{P}$ *(the auxiliary input* z *given to* $\mathcal{V}^*$ *could contain* x, y_0, y_1*).*

1.2.1 Sigma-Protocols

A Σ-protocol [26] is a 3-move interactive proof system played by two PPT algorithms $\mathcal{P}$ and $\mathcal{V}$. The algorithms receive as common input a statement "$x \in L$". $\mathcal{P}$ has as auxiliary input a witness w such that $(x, w) \in R_L$. A Σ-protocol follows the following structure: $\mathcal{P}$ speaks first sending a message a, then $\mathcal{V}$ replays with a challenge c and finally $\mathcal{P}$ sends the last message z. $\mathcal{V}$ accepts if the transcript (a, c, z) is accepting. The Σ-protocols we consider in this paper have the following properties.

Public Coin: $\mathcal{V}$ sends random bits only; i.e. $c \xleftarrow{\$} \{0,1\}^*$.

Special Soundness: let (a, c, z) and (a, c', z') be two accepting transcripts for a statement "$x \in L$". If $c \neq c'$ then $x \in L$ (i.e., the statement is true) and there exists a deterministic polynomial-time algorithm E, that on input (x, a, c, z, c', z') outputs w such that $(x, w) \in R_L$. This property implies the more general proof of knowledge property [10].

Special Honest-Verifier Zero Knowledge: there exists an efficient simulator hvSim, that on input a true statement "$x \in L$", **for any** c, produces conversations (a, c, z) distributed as indistinguishable probability distribution as conversations between the honest $\mathcal{P}$ and $\mathcal{V}$ with common input x and challenge c. Moreover, the triple (a, c, z) generated so far is an accepting transcript independently of the truthfulness of the statement (i.e., it holds also for any $x \notin L$).

Special Witness Use: knowledge of the witness is required only in the third step, i.e., in computing the message z.

Witness Indistinguishability: by Proposition 1 of [26] it holds that any three-round public-coin proof of knowledge that is honest verifier zero-knowledge, is also witness indistinguishable.

The literature provides many Σ-protocols enjoying the above properties, most notably, the protocol of Blum [12] that is a Σ-protocol for the **NP**-complete language Hamiltonicity and the protocol of Schnorr [96] for proving knowledge of a discrete logarithm. Blum's protocol is presented below in Fig. 1.1. Finally, we say that a Σ-protocol is *perfect* if it is perfect honest-verifier zero knowledge (and therefore by Proposition 1 of [26], perfectly witness indistinguishable).

Blum's protocol.

In Figure 1.1 we describe the 3-round $\mathcal{WI}\mathsf{PoK}$ protocol for the **NP**-complete language Graph Hamiltonicity (HC). We use this proof system as sub-protocol in our construction and we refer to it as BL protocol. Notice that by getting the answer for both $b = 0$ and $b = 1$ allows the extraction of the cycle. The reason is the following. For $b = 0$ one gets the permutation of the original graph G. Then for $b = 1$ one gets the of the Hamiltonian cycle of the permuted graph.

Efficient Σ-Protocols

Here we provide a list of very efficient Σ-protocol.

Σ-protocol to prove Knowledge of a Discrete Logarithm (DLog). Let $\mathbb{G}$ be a group of prime order q generated by g and let $w \in \mathbb{Z}_q$. Let n be the security parameter, and $2^n < q$. Let $\mathcal{R}_{DL}$ be the relation for the Discrete Log (DLog) problem as follows: $\mathcal{R}_{DL} = \{((\mathbb{G}, q, g, h), w) | h = g^w\}$. The Σ-protocol for $\mathcal{R}_{DL}$ is depicted in Fig. 1.2.

Σ-protocol to prove Knowledge of an Opening of a Pedersen Commitment. Let $(\mathbb{G}, p, q, g, h)$ be parameters for Pedersen commitment. Let $c = g^m h^r \in \mathbb{G}$ be the Pedersen commitment of the message $m \in \mathbb{G}$ computed with randomness $r \in \mathbb{G}$. Let $\mathcal{R}_{\mathsf{Ped}} = \{((\mathbb{G}, p, q, g, h, c), (m, r)) | c = g^m h^r \in \mathbb{G}\}$ the relation for the knowledge of a committed value. Fig. 1.3 depicts the Σ-protocol for such relation denoted by Σ_{Ped}.

Σ-protocol to prove Knowledge of the Same Opening for Two Pedersen Commitments. Let $(\mathbb{G}, p, q, g_1, h_1)$ and $(\mathbb{G}, p, q, g_2, h_2)$ be parameters for Pedersen commitment. Let $c_1 = g_1^m h_1^{r_1} \in \mathbb{G}$ and $c_2 = g_2^m h_2^{r_2} \in \mathbb{G}$ be two Pedersen commitments of the message $m \in \mathbb{Z}_q$ computed with randomness $0 < r_1, r_2 < q$. Let $\mathcal{R}_{\mathsf{Equal}}$ be the relation for the knowledge of a committed value $\mathcal{R}_{\mathsf{Equal}} = \{((\mathbb{G}, q, p, g_1, h_1, g_2, h_2, c_1, c_2), (m, r_1, r_2)) \mid c_1 = g_1^m h_1^{r_1} \in \mathbb{G}\}$ AND $c_2 = g_2^m h_2^{r_2} \in \mathbb{G}\}$. Fig. 1.4 depicts the Σ-protocol for relation $\mathcal{R}_{\mathsf{Equal}}$ denoted by Σ_{Equal}.

The BL Protocol [12] for HC.

Inputs: $\mathcal{V}$, $\mathcal{P}$ have as input a graph G. $\mathcal{P}$ has as auxiliary input a witness $y \in \mathcal{R}_{\mathsf{HC}}(G)$. Let n be the number of vertexes of G. G is represented by a $n \times n$ adjacency matrix M where $\mathsf{M}[i][j] = 1$ if there exists an edge between vertexes i and j in G, and $\mathsf{M}[i][j] = 0$ otherwise. Each of the following step is repeated **in parallel** n **times** using independent randomness.

BL1 ($\mathcal{P} \rightarrow \mathcal{V}$)**:** $\mathcal{P}$ picks a random permutation π of the graph G and commits bit-by-bit to the corresponding adjacency matrix using a statistically binding commitment scheme.

BL2 ($\mathcal{V} \rightarrow \mathcal{P}$)**:** $\mathcal{V}$ responds with a randomly chosen bit b.

BL3 ($\mathcal{P} \rightarrow \mathcal{V}$)**:**

- if $b = 0$, $\mathcal{P}$ opens all the commitments, and sends the permutation π showing that the matrix committed in step BL1 is actually the instance G.
- if $b = 1$, $\mathcal{P}$ opens only the commitments that form an Hamiltonian cycle in the permuted matrix committed in step BL1.

$\mathcal{V}$ accepts if and only if all n executions are accepting.

Figure 1.1: *Blum's witness indistinguishable proof of knowledge for Hamiltonicity.*

Common input: $x = (\mathbb{G}, q, g, h)$.
$\mathcal{P}$'s secret input: w s.t. $(x, w) \in \mathcal{R}_{DL}$.
Σ-protocol:

$\mathcal{P}$: choose $r \xleftarrow{\$} \mathbb{Z}_q$ and send $a = g^r$ to $\mathcal{V}$.
$\mathcal{V}$: choose $e \xleftarrow{\$} \mathbb{Z}_q$ and send it to $\mathcal{P}$.
$\mathcal{P}$: compute $z \leftarrow r + ew \pmod q$ and send it to $\mathcal{V}$.
$\mathcal{V}$: accept if and only if $g^z = ah^e$.

Figure 1.2: *Σ-protocol for Relation $\mathcal{R}_{DL}$.*

Proof of Knowledge of a Compound Statement. Let (x_0, x_1) be a pair of statements. $\mathcal{P}$ wants to prove that he knows a witness w such that either $(x_0, w) \in \mathcal{R}_0$ or $(x_1, w) \in \mathcal{R}_1$ without revealing which is the case. Let π_0 be the Σ-protocol for the relation $\mathcal{R}_0$ and π_1 the one for $\mathcal{R}_1$. Fig. 1.5 shows a Σ-protocol for $\mathcal{R}_0$ OR $\mathcal{R}_1$.

1.3 Universal Composability Framework

The Universal Composability framework was introduced by Canetti in [16].

In such framework, the security definition follows the standard paradigm of compar-

Common input: $x = (\mathbb{G}, p, q, g, h, c)$.
$\mathcal{P}$'s secret input: $w = (m, r)$ s.t. $c = g^m h^r \in \mathbb{G}$.
Σ-protocol:
$\mathcal{P}$: choose $t, s \xleftarrow{\$} \mathbb{Z}_q$ and send $a = g^t h^s \pmod p$ to $\mathcal{V}$.
$\mathcal{V}$: choose $e \xleftarrow{\$} \mathbb{Z}_q$ and send it to $\mathcal{P}$.
$\mathcal{P}$: compute $z_1 \leftarrow t + em$ and $z_2 \leftarrow s + er$ and send them to $\mathcal{V}$.
$\mathcal{V}$: accept if and only if $g_1^z h^{z_2} \pmod p = ac^e$.

Figure 1.3: *Protocol Σ_{Ped} for Relation $\mathcal{R}_{\mathsf{Ped}}$.*

Common input: $x = (\mathbb{G}, p, q, g_1, h_1, g_2, h_2, c_1, c_2)$.
$\mathcal{P}$'s secret input: $w = (m, r_1, r_2)$ s.t. $c_1 = g_1^m h_1^{r_1} \in G$ and $c_2 = g_2^m h_2^{r_2}$
Σ-protocol:
$\mathcal{P}$: choose $t, s \xleftarrow{\$} \mathbb{Z}_q$ and send $a_1 = g_1^t h_1^s \pmod p$, $a_2 = g_2^t h_2^s \pmod p$ to $\mathcal{V}$.
$\mathcal{V}$: choose $e \xleftarrow{\$} \{0,1\}^t$ s.t. $2^t < q$ and send it to $\mathcal{P}$.
$\mathcal{P}$: compute $z \leftarrow t + em$ and $z_1 \leftarrow s + er_1$ $z_2 \leftarrow s + er_2$ and send them to $\mathcal{V}$.
$\mathcal{V}$: accept if and only if $g_1^z h_1^{z_1} \pmod p = a_1 c_1^e$ and $g_2^z h_2^{z_2} = a_2 c_2^e \pmod p$.

Figure 1.4: *Protocol Σ_{Equal} for Relation $\mathcal{R}_{\mathsf{Equal}}$.*

Common input: (x_0, x_1).
$\mathcal{P}$'s secret input: $\mathcal{P}$ has a pair $(x_b, w) \in \mathcal{R}_b$.
Σ-protocol:
$\mathcal{P}$: compute a_b according to π_0 using (x_b, w) as input: then choose e_{1-b} and run the simulator hvSim for π_{1-b} on input (x_{1-b}, e_{1-b}) to obtain $(a_{1-b}, e_{1-b}, z_{1-b})$; send (a_0, a_1) to $\mathcal{V}$.
$\mathcal{V}$: choose s and send it to $\mathcal{P}$.
$\mathcal{P}$: set $e_b \leftarrow s \oplus e_{1-b}$, compute z according to π_b and send (e_0, z_0, e_1, z_1) to $\mathcal{V}$.
$\mathcal{V}$: check that $e_0 \oplus e_1 = s$ and that both transcript (x_0, a_0, e_0, z_0) and (x_1, a_1, e_1, z_1) are accepting according to π_0 and π_1 respectively.

Figure 1.5: *Witness-Indistinguishable PoK of a Compound Statement.*

ing a real protocol execution with an ideal protocol execution where a trusted third party (the ideal functionality) helps the participants to carry out the computation. A protocol is secure if one can show that any attack that can be mounted in the real protocol can be replicated in the ideal world experiment as well. This is formalized by requiring that for any real world adversary $\mathcal{A}$ there exists an ideal world adversary sim that is able to replicate the same attack of $\mathcal{A}$ only by interacting with the ideal functionality. In the UC setting an additional adversary $\mathbb{Z}$ called environment is introduced. The environment captures an adversary that controls $\mathcal{A}$ and concurrently interacts with other instances

of the same protocol or of different protocols and can use such interaction adaptively to help $\mathcal{A}$. Furthermore, $\mathbb{Z}$ generates the inputs for all parties, reads all outputs and interacts with the adversary $\mathcal{A}$ in arbitrary way throughout the computation.

For simplicity, we define the two-party protocol syntax, and then informally review the two-party UC-framework, which can be extended to the multi-party case. For more details, see [16].

Protocol syntax. Following [52] and [47], a protocol is represented as a system of probabilistic interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication tapes model messages sent to and received from the network. Adversarial entities are also modeled as ITMs.

The construction of a protocol in the UC-framework proceeds as follows: first, an *ideal functionality* is defined, which is a "trusted party" that is guaranteed to accurately capture the desired functionality. Then, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized.

An ideal functionality $\mathcal{F}$ is specified as an interactive Turing machine that privately communicates with the parties and the adversary and computes a task in a trusted manner. The specification of the functionality also models the adversary's ability to obtain leaked information and/or to influence the computation, also in case the adversary corrupts parties. The world in which parties privately interact with the trusted machine $\mathcal{F}$ is called ideal world.

A real protocol Π is specified as an ITM *executed by* the parties. Parties communicate over the channel in presence of an adversary $\mathcal{A}$ which controls the schedule of the communication over the channel, and can corrupt parties. When a party is corrupted the adversary receives its secret input and its internal state. In this work, we consider only *static* adversaries, which means that $\mathcal{A}$ can corrupt a party only before the protocol execution starts. This is called real world.

A protocol Π securely realizes $\mathcal{F}$ if for any real world adversary $\mathcal{A}$, there exists an ideal adversary Sim, such that the view generate by $\mathcal{A}$ running the actual protocol is indistinguishable from the view generated by Sim who has only access to the trusted party $\mathcal{F}$.

We also consider a $\mathcal{G}$-*hybrid model*, where the real-world parties are additionally given access to an ideal functionality $\mathcal{G}$. During the execution of the protocol, the parties can send inputs to, and receive outputs from, the functionality $\mathcal{G}$.

In the universally composable framework [16], the distinguisher of the views is the environment z. z has the power of choosing the inputs of all the parties and guide the actions of the adversary $\mathcal{A}$ (scheduling messages, corrupting parties), who will act just as proxy overall the execution. Let $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},z}$ be the distribution ensemble that describes the environment's output in the ideal world process, and $\mathsf{REAL}^{\mathcal{G}}_{\Pi,\mathcal{A},z}$ the distribution of the environment's output in the real world process in the $\mathcal{G}$-hybrid model.

Definition 9 (UC-Realizing an Ideal Functionality)**.** *Let $\mathcal{F}$ be an ideal functionality, and Π be a PPT protocol. We say Π* ***realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model*** *if for any hybrid-model static PPT adversary $\mathcal{A}$, there exists an ideal process expected PPT adversary Sim such that for every PPT environment z, for all auxiliary information to $z \in \{0,1\}^*$ to z, it holds:*

$$\{\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},z}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*} \sim \{\mathsf{REAL}^{\mathcal{G}}_{\Pi,\mathcal{A},z}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*} \tag{1.1}$$

1.3.1 UC Composition Theorem

The crucial theorem on the UC-framework introduced by Canetti, is the Composition Theorem. In nutshell, such theorem says that if we run a UC-secure protocol remains UC-secure even if run with any other protocol.

A more formal discussion of the UC composition theorem follows. We say that a protocol ρ "UC-emulates" a protocol ϕ if there exists an adversary S such that no environment can tell whether it is interacting with ρ and the dummy adversary, or ϕ and S. That is for every environment z,

$$\mathsf{REAL}_{\rho,D,z} \sim \mathsf{REAL}_{\phi,S,z},$$

where D is a dummy adversary. Let π, ρ, ϕ be protocols, where π may call ϕ as a subroutine. By $\pi^{\rho/\phi}$ we denote the protocol which is the same as π, except that each call to ϕ is replaced by a call to ρ.

The UC composition theorem [16] states that if ρ UC emulates ϕ, then $\pi^{\rho/\phi}$ UC emulates π. The proof of the composition theorem goes as follows: we need to show that there exists an adversary S_π such that for every environment z,

$$\mathsf{REAL}_{\pi^{\rho/\phi},D,z} \sim \mathsf{REAL}_{\pi,S_\pi,z}.$$

Let S be the adversary that follows from the fact the ρ UC emulates ϕ. We now outline the construction of adversary S_π that uses S. Adversary S_π divides the messages into two parts: those that belong to an execution of ρ, and those that belong to execution of the rest of π. The adversary S_π handles the messages pertaining to ρ by passing them to an instance of the adversary S. For the messages of π that don't belong to ρ, adversary S_π simply passes them to the parties they are intended for.

To see the correctness of the above construction, assume that there exists an environment z that distinguishes between $\pi^{\rho/\phi}$ with a dummy adversary, and π with S_π as the adversary. We will construct an environment z_ρ that distinguishes between ρ with the dummy adversary and ϕ with the adversary S. This is a contradiction, as ρ UC-emulates ϕ.

The environment z_ρ internally runs an execution of z, S_π and all the parties of π. It faithfully follows the actions of these parties except for messages concerning the sub-protocol ϕ, for which it uses the actual external parties (which are either running an instance of ϕ, or an instance of ρ). In particular, whenever an internal party of π generates an input for an instance of ρ, the environment z_ρ passes that input to the

external party. Similarly, any output sent by an external party to z_ρ is treated as an output of ϕ. Further, any message sent by S_π to a simulator S is forwarded by z_ρ to the external adversary, and the response of the external adversary is conveyed to S_π as though it was sent by S. Finally, the environment z_ρ outputs whatever z outputs.

Now, note that if the external parties are running ρ with the dummy adversary, then the view of the simulated environment z is identical to $\mathsf{REAL}_{\pi^{\rho/\phi},D,z}$. On the other hand, if the external parties are running ϕ with S, then the view of the simulated environment z is identical to $\mathsf{REAL}_{\pi,S_\pi,z}$. This implies that $\mathsf{REAL}_{\rho,D,z_\rho}$ and $\mathsf{REAL}_{\phi,S,z_\rho}$ are distinguishable, which contradicts the hypothesis that ρ UC-emulates ϕ.

On Setup Assumptions and the Hybrid Model. As proved in [21], UC-security is impossible to achieve in the plain model. Therefore, any UC-protocol must rely on some setup assumption (or must achieve some more relaxed security definition, like allowing the simulator to run in super-polynomial time). The canonical example of a setup assumption is the Common Reference String (in short, CRS). A CRS is a string that is honestly drawn from a specific distribution, and is available to the parties running an instance of the real-world protocol. The assumption is in the fact that the CRS is trusted, namely, it has been drawn honestly by a third trusted party.

In the UC-framework, a setup assumption is modeled as an ideal functionality, and the real-world protocol is augmented with access to this functionality.

An important aspect of the standard UC-framework, is that, the same instance of the setup assumption, cannot be used in more than one execution of the real-world protocol. For example, for the case of the Common Reference String, the same CRS cannot be shared by many protocol executions, but each new protocol execution requires a new, independent string. This requirements is needed for the proof of the composition theorem to go through.

The work of [17] instead consider a different formulation of the UC-framework, called generalized UC, where the same set-up assumption can be shared by many protocols. In this case we say that the setup assumption is *global*. When the set-up assumption is global, the environment has direct access to the set-up functionality, and therefore also the simulator cannot simulate the ideal functionality, but it needs to have access to it.

In [14] instead, as we shall see later, while they assume that Sim cannot simulate (*program*) a PUF, and thus has always access to the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$, they require that z has not permanent access to $\mathcal{F}_{\mathsf{PUF}}$.

In case the simulator needs access to the ideal functionality, Equation 1.1, must be rewritten as follows:

$$\{\mathsf{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathsf{Sim},z}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*} \sim \{\mathsf{REAL}^{\mathcal{G}}_{\Pi,\mathcal{A},z}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*} \tag{1.2}$$

Commitment Ideal Functionality $\mathcal{F}_{\mathsf{com}}$. The ideal functionality for Commitment Scheme as presented in [18], is depicted in Fig. 1.6.

Functionality $\mathcal{F}_{\mathsf{com}}$

$\mathcal{F}_{\mathsf{com}}$ running with parties $P_1, \ldots, P_m$ and an adversary Sim proceeds as follows:

- **Commitment Phase:** Upon receiving a message $(\mathsf{commit}, \mathsf{sid}, P_i, P_j, b)$ from P_i where $b \in \{0,1\}$, record the tuple $(\mathsf{sid}, P_i, P_j, b)$ and send the message $(\mathsf{receipt}, \mathsf{sid}, P_i, P_j)$ to P_j and Sim. Ignore any subsequent commit messages.
- **Decommit Phase:** Upon receiving $(\mathsf{open}, \mathsf{sid}, P_i, P_j)$ from P_i, if the tuple $(\mathsf{sid}, P_i, P_j, b)$ is recorded then send $(\mathsf{open}, \mathsf{sid}, P_i, P_j, b)$ to P_j and to Sim and halt. Otherwise ignore the message.

Figure 1.6: *The Commitment Functionality $\mathcal{F}_{\mathsf{com}}$.*

1.4 Additional definitions

Unconditional One-Time Message Authentication Code. A message authentication code (MAC) is defined as a triple of PPT algorithms (Gen, Mac, Vrfy). The key-generation algorithm Gen takes as input the security parameter 1^n and outputs a key k with $|k| \geq n$. The tag-generation algorithm Mac takes as input a key k and a message m and outputs a tag $t \leftarrow \mathsf{Mac}(k, m)$. The verification algorithm Vrfy takes as input a key k, a message m and a tag t and outputs 1 if t is a valid MAC of the message m, it outputs 0 otherwise. A MAC is unconditionally one-time unforgeable if, for all keys $k \leftarrow \mathsf{Gen}(1^n)$, for any adversary $\mathcal{A}$ observing a pair $(t, m) \leftarrow \mathsf{Mac}(k, m)$, probability that $\mathcal{A}$ generates a *new* pair (t', m'), such that $\mathsf{Vrfy}(k, m', t') = 1$, is negligible. Unconditional one-time MAC can be implemented using a pairwise independent function.

Definition 10 (Error correcting code)**.** *An (N, L, dis)-Error Correcting Code (ECC), is a tuple of algorithms $(\mathsf{Encode}, \mathsf{Decode})$ where $\mathsf{Encode} : \{0,1\}^N \to \{0,1\}^L$ and $\mathsf{Decode} : \{0,1\}^L \to \{0,1\}^N$ satisfy the following properties:*

Efficiency: $\mathsf{Encode}, \mathsf{Decode}$ *are deterministic polynomial time algorithms;*

Minimum Distance: $\forall m_1, m_2 \in \{0,1\}^N, \mathsf{dis}_{\mathsf{ham}}(\mathsf{Encode}(m_1), \mathsf{Encode}(m_2)) \geq \mathsf{dis}$*;*

Correct Decoding: $\forall m, \mathsf{cd} = \mathsf{Encode}(m)$, $\forall \mathsf{cd}' \in \{0,1\}^L$ *such that* $\mathsf{dis}_{\mathsf{ham}}(\mathsf{cd}, \mathsf{cd}') \leq \lfloor \frac{\mathsf{dis}}{2} \rfloor$ *it holds that* $\mathsf{Decode}(c') = m$.

In our constructions we need $(3n, L, \frac{L}{\log L})$-Error Correcting Code.

Part

UC-security from Physical Assumptions

- Chapter 2 -

Universally Composable Security from PUFs

Introduction

The impossibility of secure computation in the universal composability framework was proved first by Canetti and Fischlin [18], and then strengthened by Canetti et al. in [21]. As a consequence, several setup assumptions, and relaxations of the UC framework have been proposed to achieve UC security [22, 6, 88, 61].

In recent years, researchers have started exploring the use of secure hardware in protocol design. The idea is to achieve protocols with strong security guarantees (like UC) by allowing parties to use hardware boxes that have certain security properties. An example of the kind of security required from such a hardware box is that of *tamper-proofness*; i.e., the receiver of the box can only observe the input/output behaviour of the functionality that the box implements. This property was formalized by Katz in [62], and it was shown that UC security is possible by relying on the existence of tamper-proof programmable hardware tokens, and computational assumptions. Smart cards are well understood examples of such tokens, since they have been used in practice in the last decades. Several improvements and variations of Katz's model have been then proposed in follow up papers (e.g., [23, 75, 49, 54, 36, 25, 37]).

Spurred by technological advances in manufacturing, recently a new hardware component has gained a lot of attention: Physically Uncloneable Functions (PUFs) [82, 81]. A PUF is a hardware device generated through a special physical process that implements a "random" function[1] that depends upon the physical parameters of the process. These parameters can not be "controlled", and producing a clone of the device is considered infeasible. Once a PUF has been constructed, there is a physical procedure to query it, and to measure its answers. The answer of a PUF depends on the physical behavior of the PUF itself, and is assumed to be unpredictable, or to have high min-entropy. Namely, even after obtaining many challenge-response pairs, it is infeasible to predict the response to a new challenge.

[1] Technically, a PUF does not implement a function in the mathematical sense, as the same input might produce different responses.

Since their introduction by Pappu in 2001, PUFs have gained a lot of attention for cryptographic applications like anti-counterfeiting mechanisms, secure storage, RFID applications, identification and authentication protocols [97, 58, 58, 94, 41, 66]. More recently PUFs have been used for designing more advanced cryptographic primitives. In [92] Rührmair shows the first construction of Oblivious Transfer, the security proof of which is later provided in [93]. In [4], Armknecht et al. deploy PUFs for the construction of memory leakage-resilient encryption schemes. In [70] Maes et al. provide construction and implementation of PUFKY, a design for PUF-based cryptographic key generators. There exist several implementations of PUFs, often exhibiting different properties. The work of Armknecht et al. [3] formalizes the security features of physical functions in accordance to existing literature on PUFs and proposes a general security framework for physical functions. A survey on PUF implementations is given in [71]. Very recently in [63] Katzenbeisser et al. presented the first large scale evaluation of the security properties of some popular PUFs implementations (i.e., intrinsic electronic PUFs).

Modeling PUFs in the UC framework. Only very recently, Brzuska et al. [14] suggested a model for using PUFs in the UC setting that aims at abstracting real-world implementations. The unpredictability and uncloneability properties are modeled through an ideal functionality. Such functionality allows only the creation of trusted PUFs. In [14] PUFs are thought as non-PPT setup assumptions. As such, a PPT simulator cannot simulate a PUF, that is, PUFs are non-programmable. Although non-programmable, PUFs are not modeled as global setup [17]. [14] shows how to achieve unconditional UC secure Oblivious Transfer, Bit Commitment and Key Agreement with trusted PUFs.

PUFs vs tamper-proof hardware tokens. The apparent similarity of PUFs with programmable tamper-proof hardware tokens [62] vanishes immediately when one compares in detail the two physical devices. Indeed, PUFs are non-programmable and thus provide unpredictability only. Instead tokens are programmable and can run sophisticated code. Moreover, PUFs are stateless, while tokens can be stateful. When a PUF is not physically available, it is not possible to know the output of new queries it received. Instead the answer of a stateless token to a query is always known to its creator[2], since it knows the program embedded in the token. Tamper-proof tokens are realized through ad-hoc procedures that model them as black boxes, their internal content is protected from physical attacks and thus the functionalities that they implement can be accessed only through the prescribed input/output interface provided by the token designer. Instead, PUFs do not necessarily require such a hardware protection (which moreover could be in contrast with the need of running a physical procedure to query the PUF), and their design is associated to recommended procedures to generate and query a PUF, guaranteeing uncloneability and unpredictability. Finally, in contrast to

[2]This is true for stateful tokens too, provided that one knows the sequence of inputs received by the token.

tokens that correspond to PPT machines, PUFs are not simulatable since it is not clear if one can produce an (even computationally) indistinguishable distribution.

In this work we continue the line of research started by Brzuska et al. investigating more on the usability of PUFs to obtain UC-secure computation. We observe that the UC formulation of PUFs proposed by Brzuska et al. considers only *trusted* PUFs. This means that, it is assumed that an adversary is be unable to produce fake/malicious PUFs. We believe that making such assumption might be unrealistic. Given that the study of PUFs is still in its infancy, it is risky to rely on assumptions on the impossibility of the adversaries in generating PUFs adversarially.

The main contribution of this work consists in studying the security of protocols in presence of adversaries that can create malicious PUFs. We present a modification of the model of Brzuska et al. that formalizes security with respect to such stronger adversary and we give positive answers to the question of achieving universally composable secure computation with PUFs. More in details, our contributions are listed below.

Modeling malicious PUFs. We augment the UC framework so to enable the adversary to create untrusted (malicious) PUFs. But what exactly are malicious PUFs? In real life, an adversary could tamper with a PUF in such a way that the PUF loses any of its security properties. Or the adversary may introduce new behaviours; for example, the PUF may start logging its queries. To keep the treatment of malicious behaviour as general as possible, we allow the adversary to send as PUF any hardware token that meets the syntactical requirements of a PUF. Thus, an adversary is assumed to be able to even produce fake PUFs that might be stateful and programmed with malicious code. We assume that a malicious PUF however cannot interact with its creator once is sent away to another party. If this was not the case, then we are back in the standard model, where UC security is impossible to achieve has argued below.

UC secure computation is impossible when malicious PUFs can interact with their creator. The impossibility is straight forward. Consider any functionality that protects the privacy of the input of a player P_1. Comparing to the plain model (where UC is impossible), the only advantage of the simulator to extract the input of the real-world adversary P_1^*, is to read the challenge/answer pairs generated by P_1^* when using the honest PUF created by the simulator that plays on behalf of P_2. If such a simulator exists, then an adversary P_2^* can generate a malicious PUF that just plays as proxy and forwards back and forth what P_2^* wishes to play. P_2^* can locally use one more honest PUF in order to compute the answers that the (remote) malicious PUF is supposed to give. Clearly P_2^* will have a full view of all challenge/response pairs generated by honest P_1 and running the simulator's code, P_2^* will extract the input of P_1, therefore contradicting input privacy.

UC secure computation with malicious PUFs. The natural question is whether UC security can be achieved in such a much more hostile setting. We give a positive

answer to this question by constructing a *computational* UC commitment scheme in the malicious PUFs model. Our commitment scheme needs two PUFs that are transferred only once (PUFs do not go back-and-forth), at the beginning of the protocol and it requires computational assumptions. We avoid that PUFs go back-and-forth by employing a technique that requires OT. The results of Canetti, et al. [22] shows how to achieve general UC computation from computational UC commitments. Whether *unconditional* UC secure computation is possible in the malicious PUF model, is still an open problem.

Hardness assumptions with PUFs. Notice that as correctly observed in [14], since PUFs are not PPT machines, it is not clear if standard complexity-theoretic assumptions still hold in presence of PUFs. We agree with this observation. However the critical point is that even though there can exist a PUF that helps to break in polynomial time a standard complexity-theoretic assumptions, it is still unlikely that a PPT adversary can find such a PUF. Indeed a PPT machine can only generate a polynomial number of PUFs, therefore obtaining the one that allows to break complexity assumptions is an event that happens with negligible probability and thus it does not effect the concrete security of the protocols.

In light of the above discussion, only one of the following two cases is possible. 1) Standard complexity-theoretic assumptions still hold in presence of PPT adversaries that generate PUFs; in this case our construction is secure. 2) There exists a PPT adversary that can generate a PUF that breaks standard assumptions; in this case our construction is not secure, but the whole foundations of complexity-theoretic cryptography would fall down (which is quite unlikely to happen) with respect to real-world adversaries. We elaborate on this issue in Section 2.2.2.

Further details on our work. In our protocols after an execution the PUF can not be reused in another protocol execution. We explain how reuse of PUFs makes our commitment protocol insecure at the end of Section 2.3. However, this inability to reuse PUFs is not an artefact of our protocols, but is inherent in the UC formulation. In particular, the proof of the UC Composition Theorem [16] requires that different executions use independently created PUFs. Finding a formulation of security that allows reuse of PUFs (for e.g., by moving to the *Global* UC framework [17]) is an interesting open question. In the meantime, we feel using new PUFs for each execution is a cost we must bear for achieving UC.

The chapter is organized as follows. In Section 2.1 we provide the formal definition of Physically Uncloneable Functions. In Section 2.2 we first recall the UC-formulation of *trusted* PUFs due to [14], then we introduce our new formulation that instead model malicious PUFs.

In Section 2.3 we show how to achieve UC-secure computations in the malicious PUFs model, by providing a UC-commitment scheme that plugged into the CLOS [22] compiler [3], allows to UC-realize any functionality.

[3]More precisely, CLOS compiler requires the implementation of UC-secure multiple commitment scheme functionality$\mathcal{F}_{\mathsf{mcom}}$. We show that our commitment scheme can be extended to implement such functionality.

2.1 Definitions

We start by providing a mathematical definition of Physically Uncloneable Functions. The following definitions are taken from [14].

2.1.1 Physically Uncloneable Functions

A PUF is a noisy physical source of randomness. The randomness property comes from an uncontrollable manufacturing process. A PUF is evaluated with a physical stimulus, called the *challenge*, and its physical output, called the *response*, is measured. Because the processes involved are physical, the function implemented by a PUF can not necessarily be modeled as a mathematical function, neither can be considered computable in PPT. Moreover, the output of a PUF is noisy, namely, querying a PUF twice with the same challenge, could yield to different outputs.

A PUF-family $\mathcal{P}$ is a pair of (not necessarily efficient) algorithms Sample and Eval. Algorithm Sample abstracts the PUF fabrication process and works as follows. Given the security parameter in input, it outputs a PUF-index id from the PUF-family satisfying the security property (that we define soon) according to the security parameter. Algorithm Eval abstracts the PUF-evaluation process. On input a challenge q, it evaluates the PUF on q and outputs the response a. A PUF-family is parametrized by two parameters: the bound on the noisy of the answers d_{noise}, and the size of the output space rg. A PUF is assumed to satisfy the bounded noise condition, that is, when running $\mathsf{Eval}(1^n, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(n)$. Without loss of generality, we assume that the challenge space of a PUF is a full set of strings of a certain length.

Definition 11 (Physically Uncloneable Functions)**.** *Let rg denote the size of the range of the PUF responses of a PUF-family and d_{noise} denote a bound of the PUF's noise. $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ is a family of (rg, d_{noise})-PUF if it satisfies the following properties.*

Index Sampling. *Let $\mathcal{I}_n$ be an index set. On input the security parameter n, the sampling algorithm Sample outputs an index $\mathsf{id} \in \mathcal{I}_n$ following a not necessarily efficient procedure. Each $\mathsf{id} \in \mathcal{I}_n$ corresponds to a set of distributions $\mathcal{D}_{\mathsf{id}}$. For each challenge $q \in \{0,1\}^n$, $\mathcal{D}_{\mathsf{id}}$ contains a distribution $\mathcal{D}_{\mathsf{id}}(q)$ on $\{0,1\}^{rg(n)}$. $\mathcal{D}_{\mathsf{id}}$ is not necessarily an efficiently samplable distribution.*

Evaluation. *On input the tuple $(1^n, \mathsf{id}, q)$, where $q \in \{0,1\}^n$, the evaluation algorithm Eval outputs a response $a \in \{0,1\}^{rg(n)}$ according to distribution $\mathcal{D}_{\mathsf{id}}(q)$. It is not required that Eval is a PPT algorithm.*

Bounded Noise. *For all indexes $\mathsf{id} \in \mathcal{I}_n$, for all challenges $q \in \{0,1\}^n$, when running $\mathsf{Eval}(1^n, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(n)$.*

In this thesis we use $\mathsf{PUF}_{\mathsf{id}}(q)$ to denote $\mathcal{D}_{\mathsf{id}}(q)$. When not misleading, we omit id from $\mathsf{PUF}_{\mathsf{id}}$, using only the notation PUF.

Security Properties We assume that PUFs enjoy the properties of *uncloneability* and *unpredictability*. Unpredictability is modeled via an entropy condition on the PUF distribution. Namely, given that a PUF has been measured on a polynomial number of challenges, the response of the PUF evaluated on a new challenge has still a significant amount of entropy. In the following we recall the concept of average min-entropy, and the formal definition of PUF-unpredictability.

Definition 12 (Average min-entropy)**.** *The average min-entropy of the measurement* $\mathsf{PUF}(q)$ *conditioned on the measurements of challenges* $\mathcal{Q} = (q_1, \ldots, q_\ell)$ *is defined by:*

$$\begin{aligned}\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) &= -log\big(\mathbb{E}_{a_i \leftarrow \mathsf{PUF}(q_i)}[\max_a \mathsf{Pr}\,[\,\mathsf{PUF}(q) = a | a_1 = \mathsf{PUF}(q_1), \ldots, a_\ell = \mathsf{PUF}(q_\ell)\,]\,\big) \\ &= -log\big(\mathbb{E}_{a_i \leftarrow \mathsf{PUF}(q_i)}[2^{H_\infty(\mathsf{PUF}(q)=a|a_1=\mathsf{PUF}(q_1),\ldots,a_\ell=\mathsf{PUF}(q_\ell)}\big)\end{aligned}$$

where the probability is taken over the choice of id *from the PUF-family and the choice of possible PUF responses on challenge* q*. The term* $\mathsf{PUF}(\mathcal{Q})$ *denotes a sequence of random variables* $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_\ell)$ *each corresponding to an evaluation of the PUF on challenge* q_k*.*

Definition 13 (Unpredictability)**.** *A* (rg, d_{noise})*-PUF family* $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ *for security parameter* n *is* $(d_{\mathsf{min}}(n), \tilde{m}(n))$*-unpredictable if for any* $q \in \{0,1\}^n$ *and challenge list* $\mathcal{Q} = (q_1, \ldots, q_n)$*, one has that, if for all* $1 \leq k \leq n$ *the Hamming distance satisfies* $\mathsf{dis}_{\mathsf{ham}}(q, q_k) \geq d_{\mathsf{min}}(n)$*, then the average min-entropy satisfies* $\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) \geq m(n)$*. Such a PUF-family is called a* $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$*- PUF family.*

The above unpredictability definition automatically implies uncloneability (see [15] pag. 39 for details).

2.1.2 Fuzzy Extractors

The output of a PUF is noisy. That is, querying the PUF twice with the same challenge two times, one might obtain distinct responses a, a', that are at most d_{noise} apart in hamming distance. Fuzzy extractors of Dodis et al. [35] are applied to the outputs of the PUF, to convert such noisy, high-entropy measurements into *reproducible* randomness. Very roughly, a fuzzy extractor is a pair of efficient randomized algorithms $(\mathsf{FuzGen}, \mathsf{FuzRep})$, and it is applied to PUFs 'responses as follows. FuzGen takes as input an ℓ-bit string, that is the PUF's response a, and outputs a pair (p, st), where st is a uniformly distributed string, and p is a public helper data string. FuzRep takes as input the PUF's noisy response σ' and the helper data p and generates the very same string st obtained with the original measurement a.

The security property of fuzzy extractors guarantees that, if the min-entropy of the PUF's responses are greater than a certain parameter m, knowledge of the public data p only, without the measurement a, does not give any information on the secret value st. The correctness property, guarantees that, all pairs of responses σ, σ' that are close enough, i.e., their hamming distance is less then a certain parameter t, will be recovered by FuzRep to the same value st generated by FuzGen. Clearly, in order to apply fuzzy

extractors to PUF's answers, it is sufficient to pick an extractor whose parameters match with the parameter of the PUF being used.

Let U_ℓ denote the uniform distribution on ℓ-bit strings. Let $\mathcal{M}$ be a metric space with the distance function dis: $\mathcal{M} \times \mathcal{M} \to \mathbb{R}^+$.

Definition 14 (Fuzzy Extractors)**.** *Let* dis *be a distance function for metric space* $\mathcal{M}$*. A* (m, ℓ, t, ϵ)*-fuzzy extractor is a pair of efficient randomized algorithms* $(\mathsf{FuzGen}, \mathsf{FuzRep})$*. The algorithm* FuzGen *on input* $w \in \mathcal{M}$*, outputs a pair* (p, st)*, where* $st \in \{0,1\}^\ell$ *is a secret string and* $p \in \{0,1\}^*$ *is a helper data string. The algorithm* FuzRep*, on input an element* $w' \in \mathcal{M}$ *and a helper data string* $p \in \{0,1\}^*$ *outputs a string* st*. A fuzzy extractor satisfies the following properties.*

Correctness. *For all* $w, w' \in \mathcal{M}$*, if* $\mathsf{dis}(w, w') \leq t$ *and* $(st, p) \xleftarrow{\$} \mathsf{FuzGen}$*, then* $\mathsf{FuzRep}(w', p) = st$*.*

Security. *For any distribution* $\mathcal{W}$ *on the metric space* $\mathcal{M}$*, that has min-entropy* m*, the first component of the random variable* (st, p)*, defined by drawing* w *according to* $\mathcal{W}$ *and then applying* FuzGen*, is distributed almost uniformly, even given* p*, i.e.,* $SD((st, p), (U_\ell, p)) \leq \epsilon$*.*

Given a $(rg(n), d_{\mathsf{noise}}(n), d_{\mathsf{min}}(n), m(n))$-PUF family with $d_{\mathsf{min}}(n) = o(n/\log n)$, a *matching* fuzzy extractor has as parameters $\ell(n) = n$ and $t(n) = d_{\mathsf{noise}}(n)$. The metric space $\mathcal{M}$ is the range $\{0,1\}^{rg}$ with Hamming distance $\mathsf{dis}_{\mathsf{ham}}$. We call such PUF family and fuzzy extractor as having matching parameters, and the following properties are guaranteed.

Well-Spread Domain. For all polynomial $p(n)$ and all set of challenges $q_1, \ldots, q_{p(n)}$, the probability that a randomly chosen challenge is within distance smaller than d_{min} with any q_k, for $1 \leq k \leq n$ is negligible.

Extraction Independence. For all challenges $q_1, \ldots, q_{p(n)}$, and for a challenge q such that $\mathsf{dis}(q, q_k) > d_{\mathsf{min}}$ for $1 \leq k \leq p(n)$, it holds that the PUF evaluation on q and subsequent application of FuzGen yields an almost uniform value st even if p is observed.

Response consistency. Let a, a' be the responses of PUF when queried twice with the same challenge q, then for $(st, p) \xleftarrow{\$} \mathsf{FuzGen}(a)$ it holds that $st \leftarrow \mathsf{FuzRep}(a', p)$.

2.2 PUFs in the UC framework

In this section we first present the original UC-formulation of honest PUFs proposed by Brzuska et. al [14]. Then we show how we extend it in order to allow the adversary to create and use malicious PUFs.

2.2.1 Modeling Trusted PUFs

As mentioned before, the model of Brzuska et. al. considers only *trusted* PUFs. This manifests from the definition of the ideal functionality of PUFs, that we denote by $\mathcal{F}_{\mathsf{hPUF}}$ and we formally describe in Fig. 2.1. Indeed, consider $\mathcal{F}_{\mathsf{hPUF}}$. To create a PUF, a party has to send the command $\mathsf{init}_{\mathsf{PUF}}$ to $\mathcal{F}_{\mathsf{hPUF}}$. Then the functionality will run the honest procedure to generate a PUF (namely, it runs the Sample algorithm). Therefore, even a corrupted party clearly cannot interfere in the generation of the PUF.

2.2.2 Modeling Malicious PUFs

We allow our adversaries to send malicious PUFs to honest parties. As discussed before, the motivation for malicious PUFs is that the adversary may have some control over the manufacturing process and may be able to produce errors in the process that break the PUF's security properties. Thus, we would like parties to rely on only the PUFs that they themselves manufacture (or obtain from a source that they trust), and not on the ones they receive from other (possibly adversarial) parties.

Malicious PUF Families. In the real world, an adversary may create a malicious PUF in a number of ways. For example, it can tamper with the manufacturing process for an honestly-generated PUF to compromise its security properties (unpredictability, for instance). It may also introduce additional behaviour into the PUF token, like logging of queries. Taking inspiration from the literature on modeling tamper-proof hardware tokens, one might be tempted to model malicious PUFs analogously in the following way: to create a malicious PUF, the adversary simply specifies to the ideal functionality, the (malicious) code it wants to be executed instead of an honest PUF. Allowing the adversary to specify the malicious code enables the simulator to "rewind" the malicious PUF, which is used crucially in security proofs in the hardware token model. However, modeling malicious PUFs in this way would disallow the adversary from modifying honest PUFs (or more precisely, the honest PUF manufacturing process). To keep our treatment as general as possible, we do not place any restriction on a malicious PUF, except that it should have the same syntax as that of an honest PUF family, as specified in Denition 11. In particular, the adversary is not required to know the code of malicious PUFs it creates, and thus our simulator can not rely on rewinding in the security proofs. Formally, we allow the adversary to specify a "malicious PUF family", that the ideal functionality uses. Of course, in the protocol, we also want the honest parties to be able to obtain and send honestly generated PUFs. Thus our ideal functionality for PUFs, $\mathcal{F}_{\mathsf{PUF}}$ (Fig. 2.2) is parameterized by two PUF families: the normal (or honest) family $(\mathsf{Sample}_{\mathtt{normal}}, \mathsf{Eval}_{\mathtt{normal}})$ and the possibly malicious family $(\mathsf{Sample}_{\mathtt{mal}}, \mathsf{Eval}_{\mathtt{mal}})$. When a party P_i wants to initialize a PUF, it sends a $\mathsf{init}_{\mathsf{PUF}}$ message to $\mathcal{F}_{\mathsf{PUF}}$ in which it specifies the $\mathtt{mode} \in \{\mathtt{normal}, \mathtt{mal}\}$, and the ideal functionality uses the corresponding family for initializing the PUF. For each initialized PUF, the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ also stores a tag representing the family (i.e., $\mathtt{mal}$ or $\mathtt{normal}$) from which it was initialized. Thus, when the PUF needs to be evaluated, $\mathcal{F}_{\mathsf{PUF}}$ runs the evaluation algorithm corresponding to the tag.

$\mathcal{F}_{\mathsf{PUF}}(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$ receives as initial input a security parameter 1^n and runs with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$.

- When a party P_i writes $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$, then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathsf{sid}, *, *, *, *)$:
  - If this is the case, then turn into the waiting state.
  - Else, draw $\mathsf{id} \leftarrow \mathsf{Sample}(1^n)$ from the PUF-family. Put $(\mathsf{sid}, \mathsf{id}, P_i, *, \mathsf{notrans})$ in $\mathcal{L}$ and write $(\mathsf{initialized}_{\mathsf{PUF}}, \mathsf{sid})$ on the communication tape of P_i.
- When party P_i writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, P_i, q)$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape, check if there exists a tuple $(\mathsf{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}(1^n, \mathsf{id}, q)$. Write $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ on P_i's communication input tape.
- When a party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if there exists a tuple $(\mathsf{sid}, *, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, modify the tuple $(\mathsf{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ to the updated tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$. Write $(\mathsf{invoke}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ on $\mathcal{S}$'s communication input tape.
- When the adversary sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S}, q)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains a tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(*))$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}(1^n, \mathsf{id}, q)$ and return $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ to $\mathcal{S}$.
- When $\mathcal{S}$ sends $(\mathsf{ready}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S})$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains the tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$.
  - If not found, turn into the waiting state.
  - Else, change the tuple $(\mathsf{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$ to $(\mathsf{sid}, \mathsf{id}, P_j, \mathsf{notrans})$ and write $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on P_j's communication input tape and store the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$.
- When the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ exists in L. If not, return to the waiting state. Else, write this tuple to the communication input tape of P_i.

Figure 2.1: *The ideal functionality $\mathcal{F}_{\mathsf{hPUF}}$ from Brzuska et.al. [14] .*

As in the original formulation of Brzuska et al., the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ keeps a list $\mathcal{L}$ of tuples $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \hat{P}, \tau)$. Here, sid is the session identifier of the protocol and id is the PUF identifier output by the $\mathsf{Sample}_{\mathtt{mode}}$ algorithm. As discussed above $\mathtt{mode} \in \{\mathtt{normal}, \mathtt{mal}\}$ indicates the mode of the PUF, and $\hat{P}$ identifies the party that currently holds the PUF. The final argument τ specifies transition of PUFs: $\tau = \mathsf{notrans}$ indicates the PUF is not in transition, while $\tau = \mathsf{trans}(P_j)$ indicates that the PUF is in transition to party P_j. Only the adversary may query the PUF during the transition period. Thus, when a party P_i hands over a PUF to party P_j, the corresponding τ value for that PUF is changed from $\mathsf{notrans}$ to $\mathsf{trans}(P_j)$, and the adversary is allowed to send evaluation queries to this PUF. When the adversary is done with querying the PUF, it sends a $\mathsf{ready}_{\mathsf{PUF}}$ message to the ideal functionality, which hands over the PUF to P_j and changes the PUFs transit flag back to $\mathsf{notrans}$. The party P_j may now query the PUF. The ideal functionality now waits for a $\mathsf{received}_{\mathsf{PUF}}$ message from the adversary, at which point it sends a $\mathsf{received}_{\mathsf{PUF}}$ message to P_i informing it that the hand over is complete. The ideal functionality is described formally in Fig. 2.2.

Allowing Adversary to Create PUFs. We deviate from the original formulation of $\mathcal{F}_{\mathsf{PUF}}$ of Brzuska et al. [14] in one crucial way: we allow the ideal-world adversary $\mathcal{S}$ to create new PUFs. That is, $\mathcal{S}$ can send a $\mathsf{init}_{\mathsf{PUF}}$ message to $\mathcal{F}_{\mathsf{PUF}}$. In the original formulation of Brzuska et al., $\mathcal{S}$ could not create its own PUFs, and this has serious implications for the composition theorem, as it was discovered by Vald [98]. We elaborate on this issue in a separate paragraph at the end of the section. Also, it should be noted that the PUF set-up is non-programmable, but not *global* [17]. The environment must go via the adversary to query PUFs, and may only query PUFs in transit or held by the adversary at that time.

We remark that the OT protocol of [14] for honest PUFs, fails in the presence of malicious PUFs. Consider the OT protocol in Fig. 3 in [14]. The security crucially relies on the fact that the receiver P_j *can not* query the PUF after receiving sender's first message, i.e., the pair (x_0, x_1). If it could do so, then it would query the PUF on both $x_0 \oplus v$ and $x_1 \oplus v$ and learn both s_0 and s_1. In the malicious PUF model however, as there is no guarantee that the receiver can not learn query/answer pairs when a malicious PUF that he created is not in its hands, the protocol no longer remains secure.

PUFs and computational assumptions. The protocol we present in the next section will use computational hardness assumptions. These assumptions hold against probabilistic polynomial-time adversaries. However, PUFs use physical components and are not modeled as PPT machines, and thus, the computational assumptions must additionally be secure against PPT adversaries that have access to PUFs. We remark that this is a reasonable assumption to make, as if this is not the case, then PUFs can be used to invert one-way functions, to find collisions in CRHFs and so on, therefore not only our protocol, but any computational-complexity based protocol would be insecure. Note that PUFs are physical devices that actually exist in the real world, and thus all real-world adversaries could use them.

To formalize this, we define the notion of "admissible" PUF families. Informally, a PUF family (regardless of whether it is honest or malicious) is called *admissible* with

$\mathcal{F}_{\mathsf{PUF}}$ uses PUF families $\mathcal{P}_1 = (\mathsf{Sample}_{\mathtt{normal}}, \mathsf{Eval}_{\mathtt{normal}})$ with parameters $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$, and $\mathcal{P}_2 = (\mathsf{Sample}_{\mathtt{mal}}, \mathsf{Eval}_{\mathtt{mal}})$. It runs on input the security parameter 1^n, with parties $\mathbb{P} = \{ P_1, \ldots, P_n \}$ and adversary $\mathcal{S}$.

- When a party $\hat{P} \in \mathbb{P} \cup \{ \mathcal{S} \}$ writes $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, \mathtt{mode}, \hat{P})$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$, where $\mathtt{mode} \in \{ \mathtt{normal}, \mathtt{mal} \}$, then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathsf{sid}, *, *, *, *)$:

  If this is the case, turn in waiting state. **Else**, draw $\mathsf{id} \leftarrow \mathsf{Sample}_{\mathtt{mode}}(1^n)$ from the PUF family. Put $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \hat{P}, \mathsf{notrans})$ in $\mathcal{L}$ and write $(\mathsf{initialized}_{\mathsf{PUF}}, \mathsf{sid})$ on the communication tape of $\hat{P}$.

- When party $P_i \in \mathbb{P}$ writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, P_i, q)$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape, check if there exists a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ in $\mathcal{L}$.

  If not, turn in waiting state. **Else**, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, q)$. Write $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ on P_i's input tape.

- When a party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if there exists a tuple $(\mathsf{sid}, *, *, P_i, \mathsf{notrans})$ in $\mathcal{L}$.

  If not, turn in waiting state. **Else**, modify the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ to the updated tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$. Write $(\mathsf{invoke}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ on $\mathcal{S}$'s input tape.

- When the adversary sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S}, q)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(*))$ or $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \mathcal{S}, \mathsf{notrans})$.

  If not, turn in waiting state. **Else**, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, q)$ and return $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ to $\mathcal{S}$.

- When $\mathcal{S}$ sends $(\mathsf{ready}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S})$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$.

  If not found, turn in waiting state. **Else**, change the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$ to $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_j, \mathsf{notrans})$ and write $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on P_j's input tape and store the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$.

- When the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ has been stored. If not, return to the waiting state. Else, write this tuple to the input tape of P_i.

Figure 2.2: *The ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ for malicious PUFs.*

respect to a hardness assumption if that assumption holds even when the adversary has access to PUFs from this family. We will prove that our protocol is secure when the $\mathcal{F}_{\mathsf{PUF}}$ ideal functionality is instantiated with admissible PUF families.

For our purpose, all the cryptographic tools that we use to construct our protocols can be based on the DDH assumption. Thus, we define PUF families that are admissible only with respect to DDH, but note that the definition can be generalized to other cryptographic primitives. This is a straightforward generalization of the standard DDH definition. The following formulation is adapted from [86].

Let G be an algorithm that takes as input a security parameter n and outputs a tuple $\mathbb{G} = (p, \langle G \rangle, g)$, where p is a prime, $\langle G \rangle$ is the description of a cyclic multiplicative group G of order p, and g is a generator of G.

Definition 15. *Let* G *be as defined above, and let* $(\mathsf{Sample}, \mathsf{Eval})$ *be a PUF family. The tuple* $(\mathsf{G}, (\mathsf{Sample}, \mathsf{Eval}))$ *is called* DDH-admissible *if for every oracle PPT distinguisher* D*, for every polynomial* $p(\cdot)$*, and for sufficiently large* n*,*

$$\left| \mathsf{Pr}\left[D^{\mathsf{Sample}(\cdot),\mathsf{Eval}(\cdot)}(1^n, (\mathbb{G}, g^a, g^b, g^{ab})) = 1 \right] - \mathsf{Pr}\left[D^{\mathsf{Sample}(\cdot),\mathsf{Eval}(\cdot)}(1^n, (\mathbb{G}, g^a, g^b, g^c)) = 1 \right] \right| \leq \frac{1}{p(n)},$$

where $\mathbb{G} \leftarrow \mathsf{G}(1^n)$ *and* $a, b, c \leftarrow \mathbb{Z}_p$ *are uniform and independent.*

For succinctness, we will often keep G implicit in our discussion, and refer to a PUF family for which $(\mathsf{G}, (\mathsf{Sample}, \mathsf{Eval}))$ is DDH-admissible simply as "admissible".

From this point on in this paper, we only talk of admissible families.

Allowing Adversary to Create PUFs. We now explain why it is crucial for the UC-composition theorem to allow the adversary to create PUFs of its own. We begin by sketching the proof of the UC composition theorem from Section 5.2, [16]. Those familiar with the proof of the composition theorem can skip to the end of this section.

We say that a protocol ρ "UC-emulates" a protocol ϕ if there exists an adversary S such that no environment can tell whether it is interacting with ρ and the dummy adversary, or ϕ and S. That is for every environment z,

$$\mathsf{REAL}_{\rho,D,z} \sim \mathsf{REAL}_{\phi,S,z},$$

where D is a dummy adversary. Let π, ρ, ϕ be protocols, where π may call ϕ as a subroutine. By $\pi^{\rho/\phi}$ we denote the protocol which is the same as π, except that each call to ϕ is replaced by a call to ρ.

The UC composition theorem [16] states that if ρ UC emulates ϕ, then $\pi^{\rho/\phi}$ UC emulates π. The proof of the composition theorem goes as follows: we need to show that there exists an adversary S_π such that for every environment z,

$$\mathsf{REAL}_{\pi^{\rho/\phi},D,z} \sim \mathsf{REAL}_{\pi,S_\pi,z}.$$

Let S be the adversary that follows from the fact the ρ UC emulates ϕ. We now outline the construction of adversary S_π that uses S. Adversary S_π divides the messages into two parts: those that belong to an execution of ρ, and those that belong to execution of the rest of π. The adversary S_π handles the messages pertaining to ρ by passing them

to an instance of the adversary S. For the messages of π that don't belong to ρ, adversary S_π simply passes them to the parties they are intended for.

To see the correctness of the above construction, assume that there exists an environment z that distinguishes between $\pi^{\rho/\phi}$ with a dummy adversary, and π with S_π as the adversary. We will construct an environment z_ρ that distinguishes between ρ with the dummy adversary and ϕ with the adversary S. This is a contradiction, as ρ UC-emulates ϕ.

The environment z_ρ internally runs an execution of z, S_π and all the parties of π. It faithfully follows the actions of these parties except for messages concerning the sub-protocol ϕ, for which it uses the actual external parties (which are either running an instance of ϕ, or an instance of ρ). In particular, whenever an internal party of π generates an input for an instance of ρ, the environment z_ρ passes that input to the external party. Similarly, any output sent by an external party to z_ρ is treated as an output of ϕ. Further, any message sent by S_π to a simulator S is forwarded by z_ρ to the external adversary, and the response of the external adversary is conveyed to S_π as though it was sent by S. Finally, the environment z_ρ outputs whatever z outputs.

Now, note that if the external parties are running ρ with the dummy adversary, then the view of the simulated environment z is identical to $\mathsf{REAL}_{\pi^{\rho/\phi},D,z}$. On the other hand, if the external parties are running ϕ with S, then the view of the simulated environment z is identical to $\mathsf{REAL}_{\pi,S_\pi,z}$. This implies that $\mathsf{REAL}_{\rho,D,z_\rho}$ and $\mathsf{REAL}_{\phi,S,z_\rho}$ are distinguishable, which contradicts the hypothesis that ρ UC-emulates ϕ.

Composition Theorem and PUFs. Now let us consider composition in the presence of PUFs. Recall that in the $\mathcal{F}_{\mathsf{PUF}}$-hybrid model, the environment does not have direct access to the ideal $\mathcal{F}_{\mathsf{PUF}}$ functionality (see Section 4.3 and Appendix B of [14] for details on the PUF access model). However, looking at the proof of the composition theorem, we immediately notice that the environment z_ρ must have the ability to create PUFs. This is because to carry out the internal simulation of z and π, environment z_ρ must be able to handle PUF requests by the parties of π. Since PUFs are not programmable, z_ρ can not simulate PUF responses on its own. It is to tackle this very issue that we allow the adversary to create new PUF's in our $\mathcal{F}_{\mathsf{PUF}}$ ideal functionality in Figure 2.2. This is sufficient for the composition theorem: when a simulated party in π requests a PUF, the environment z_ρ asks the adversary to create a new PUF, and uses that PUF to handle the simulated party's requests.

2.3 UC Secure Computation in the Malicious PUFs model

In this section we present a construction that UC-realizes the commitment functionality $\mathcal{F}_{\mathsf{com}}$ in the malicious PUFs model, and thus UC security for any PPT functionality using the compiler of [22].

Let us recall some of the peculiarities of the PUFs model. A major difficulty when using PUFs, in contrast to tamper-proof tokens, is that PUFs are *not programmable*. That is, the simulator can not simulate the answer of a PUF, and it must honestly forward the queries to the $\mathcal{F}_{\mathsf{PUF}}$ functionality. The only power of the simulator is to

intercept the queries made by the adversary to honest PUFs. Thus, in designing the protocol, we shall force parties to query the PUFs with the critical private information related to the protocol, so as to allow the simulator to extract such information in straight-line. In the malicious PUFs model the behaviour of a PUF sent by an adversary is entirely in the adversary's control. A malicious PUF can answer (or even abort) adaptively on the query according to some pre-shared strategy with the malicious creator. Finally, a side effect of the unpredictability of PUFs, is that the creator of a honest PUF is not able to check the authenticity of the answer generated by its own PUF, without having the PUF in its hands (or having queried the PUF previously on the very same value). This might generate the undesired effect of PUFs going back and forth during the protocol.

Techniques and proof intuition. Showing UC security for commitments requires obtaining straight-line extraction against a malicious sender and straight-line equivocality against a malicious receiver. Our starting point is the equivocal commitment scheme of [27] which builds upon Naor's scheme [76], that consists of two messages. The first message is a randomly chosen string r that the receiver sends to the sender. The second message is the commitment of the bit b, computed using r. More precisely, it is $G(s) \oplus (r \wedge \mathbf{b})$, where $G()$ is a PRG, and s a randomly chosen seed and $\mathbf{b}$ is the string $b^{|r|}$. The scheme has the property that if the string r is crafted appropriately, then the commitment is equivocal. [27] shows how this can be achieved by adding a coin-tossing phase before the commitment. The coin tossing of [27] proceeds as follows: the receiver commits to a random string α (using a statistically hiding commitment scheme), the sender sends a string β, and then the receiver opens the commitment. Naor's parameter r is then set as $\alpha \oplus \beta$.

Observe that if the simulator can choose β after knowing α, then it can control the output of the coin-tossing phase, and therefore equivocate the commitment. Thus, to achieve equivocality against a malicious receiver, the simulator must be able to extract α from the commitment. Similarly, when playing against a malicious sender, the simulator should be able to extract the value committed in the second message of Naor's commitment.

Therefore, to construct a UC-secure commitment, we need to design an extractable commitment scheme for both directions. The extractable commitment of α that we construct for the receiver, must be statistically-hiding (this is necessary to prove binding). We denote such commitment as $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$. Instead, the commitment sent by the sender, must be extractable and allow for equivocation. We denote such commitment as $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$. As we shall see soon, the two schemes require different techniques as they aim to different properties. However, they both share the following structure to achieve extractability.

The receiver creates a PUF and queries it with two randomly chosen challenges (q_0, q_1), obtaining the respective answers (a_0, a_1). The PUF is then sent to the sender. To commit to a bit b, the sender first needs to obtain the value q_b. This is done by running an OT protocol with the receiver. Then the sender queries the PUF with q_b and

commits to the response a_b. Note that the sender does not commit to the bit directly, but to the *answer* of the PUF. This ensure extractability. To decommit to b, the sender simply opens the commitment of the PUF-answer sent before. Note that the receiver can check the authenticity of the PUF-answer without having its own PUF back. The simulator can extract the bit by intercepting the queries sent to the PUF and taking the one that is close enough, in Hamming distance, to either q_0 or q_1. Due to the security of OT, the sender can not get both queries (thus confusing the simulator), neither can the receiver detect which query has been transferred. Due to the binding property of the commitment scheme used to commit q_b, a malicious sender cannot postpone querying the PUF to the decommitment phase (thus preventing the simulator to extract already in the commitment phase). Due to the unpredictability of PUFs, the sender cannot avoid to query the PUF to obtain the correct response.

This protocol achieves extractability. To additionally achieve statistically hiding and equivocality, protocol $\mathsf{Com}_{\mathsf{shext}}$ and $\mathsf{Com}_{\mathsf{equiv}}$ develop on this basic structure in different ways accordingly to the different properties that they achieve. The main difference is in the commitment of the answer a_b.

In Protocol $\mathsf{Com}_{\mathsf{shext}}$, $\mathsf{C}_{\mathsf{shext}}$ commits to the PUF-response a_b using a statistically hiding commitment scheme. Additionally, $\mathsf{C}_{\mathsf{shext}}$ provides a statistical zero-knowledge argument of knowledge of the message committed. This turns out to be necessary to argue about binding (that is only computational). Finally, the OT protocol executed to exchange q_0, q_1 must be statistically secure for the OT receiver. A graphic high-level description of $\mathsf{Com}_{\mathsf{shext}}$ is provided in Fig. 2.3.

In Protocol $\mathsf{Com}_{\mathsf{equiv}}$ the answer a_b is committed following Naor's commitment scheme. The input of $\mathsf{C}_{\mathsf{equiv}}$ is the Naor's parameter decided in the coin-flipping phase, and is the vector $\bar{r}$ of strings $r_1, \ldots, r_l$ (a_b is a l-bit string where l is the range of the PUF). Earlier we said that the simulator can properly craft $\bar{r}$, so that it will be able equivocate the commitment of a_b. However, due to the structure of the extractable commitment shown above, being able to equivocate the commitment of a_b is not enough anymore. Indeed, in the protocol above, due to the OT protocol, the simulator will be able to obtain only one of the PUF-queries among (q_0, q_1), and it must choose the query q_b already in the commitment phase (when the secret bit b is not known to the simulator). Thus, even though the simulator has the power to equivocate the commitment to any string, it might not know the correct PUF-answer to open to. We solve this problem by asking the receiver to reveal both values (q_0, q_1) played in the OT protocol (along with the randomness used in the OT protocol), obviously only after $\mathsf{C}_{\mathsf{equiv}}$ has committed to the PUF-answer. Now, the simulator can: play the OT protocol with a random bit, commit to a random string (without querying the PUF), and then obtain both queries (q_0, q_1). In the decommitment phase, the simulator gets the actual bit b. Hence, it can query the PUF with input q_b, obtain the PUF-answer, and equivocate the commitment so to open to such PUF-answer. There is a subtle issue here and is the possibility of *selective abort* of a malicious PUF. If the PUF aborts when queried with a particular string, then we have that the sender would abort already in the commitment phase, while the simulator aborts only in the decommitment phase. We avoid such problem by

requiring that the sender continues the commitment phase by committing to a random string in case the PUF aborts. The above protocol is statistically binding (we are using Naor's commitment), straight-line extractable, and assuming that Naor's parameter was previously ad-hoc crafted, it is also straight-line equivocal. To commit to a bit we are committing to the l-bit PUF-answer, thus the size of Naor's parameter is $N = (3n)l$. Protocol $\mathsf{Com}_{\mathsf{equiv}}$ is formally described in Fig. 2.4.

The final UC-secure commitment scheme $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ consists of the coin-flipping phase, and the (equivocal) commitment phase. In the coin flipping, the receiver commits to α using the statistically hiding straight-line extractable commitment scheme $\mathsf{Com}_{\mathsf{shext}}$. The output of the coin-flipping is the Naor's parameter $\bar{r}{=}\alpha \oplus \beta$ used as common input for the extractable/equivocal commitment scheme $\mathsf{Com}_{\mathsf{equiv}}$. Protocol $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ is depicted in Fig. 2.6.

Both protocol $\mathsf{Com}_{\mathsf{shext}}, \mathsf{Com}_{\mathsf{equiv}}$ require one PUF sent from the receiver to the sender. We remark that PUFs are transferred only **once at the beginning of the protocol**.

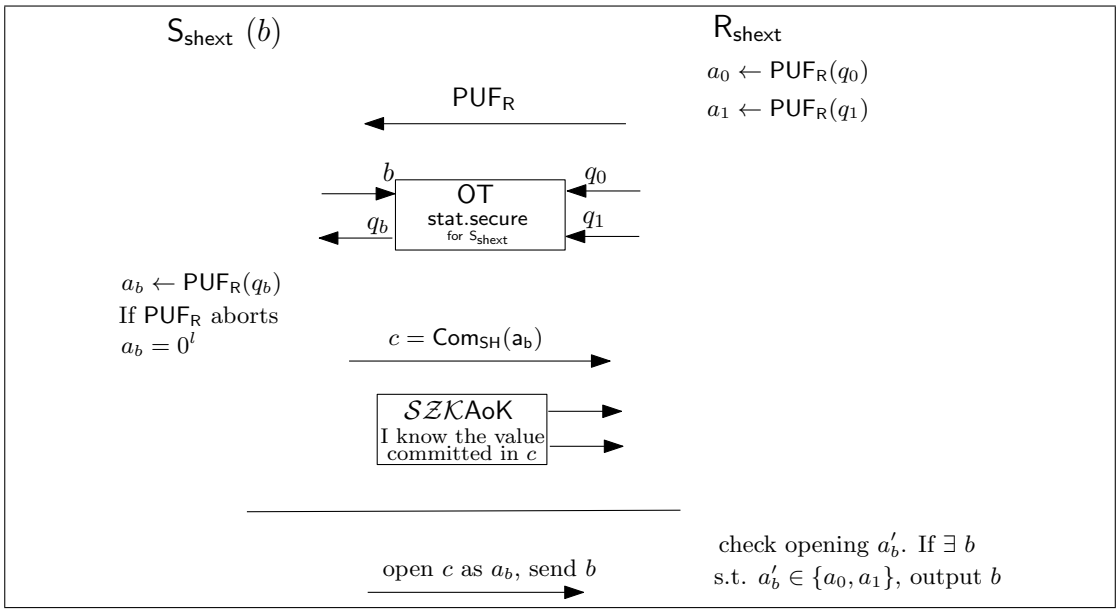

Figure 2.3: *Protocol* $\mathsf{C}_{\mathsf{shext}}$

Theorem 1. *If* $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$ *is a statistically hiding straight-line extractable commitment scheme in the malicious PUFs model, and* $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$ *is a statistically binding straight-line extractable and equivocal commitment scheme in the malicious PUFs model, then* $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ *in Fig. 2.6, UC-realizes the* $\mathcal{F}_{\mathsf{com}}$ *functionality.*

The above protocol can be used to implement the multiple commitment functionality $\mathcal{F}_{\mathsf{mcom}}$ by using independent PUFs for each commitment. Note that in our construction

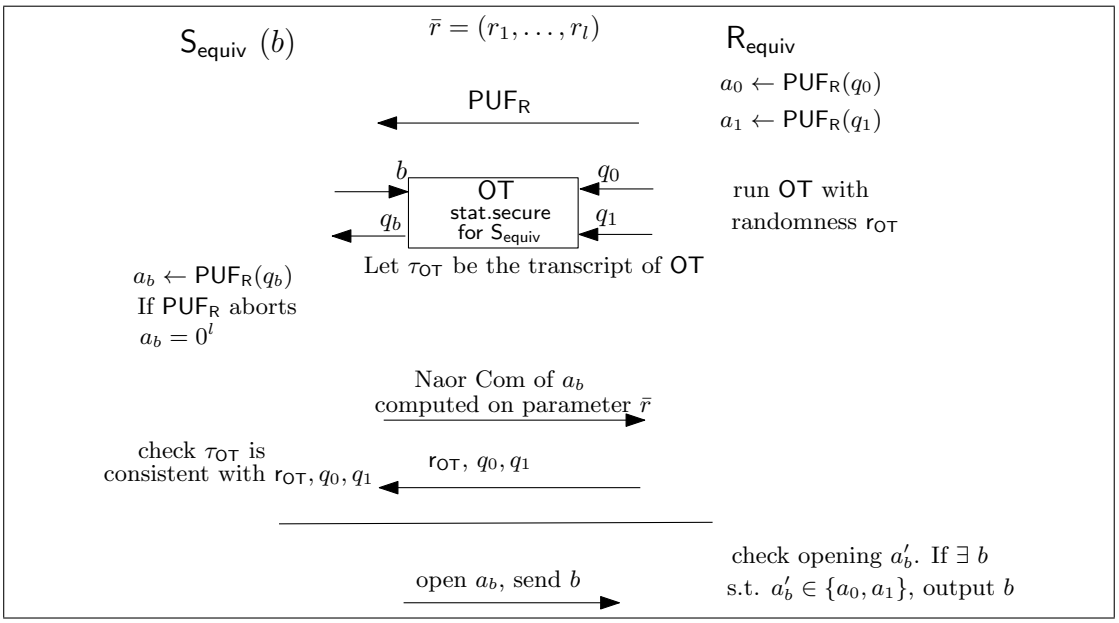

Figure 2.4: *Protocol* $\mathsf{Com}_{\mathsf{equiv}}$

we can not reuse the *same* PUF when multiple commitments are executed *concurrently*[4]. The reason is that, in both sub-protocols $\mathsf{Com}_{\mathsf{shext}}, \mathsf{Com}_{\mathsf{equiv}}$, in the opening phase the committer forwards the answer obtained by querying the receiver's PUF. The answer of a malicious PUF can then convey information about the value committed in concurrent sessions that have not been opened yet.

When implementing $\mathcal{F}_{\mathsf{mcom}}$ one should also deal with malleability issues. In particular, one should handle the case in which the man-in-the-middle adversary forwards honest PUFs to another party. However such attack can be easily ruled out by exploiting the unpredictability of honest PUFs as follows. Let P_i be the creator of PUF_i, running an execution of the protocol with P_j. Before delivering its own PUF, P_i queries it with the identity of P_j concatenated with a random *nonce*. Then, at some point during the protocol execution with P_j it will ask P_j to evaluate PUF_i on such *nonce* (and the identity). Due to the unpredictability of PUFs, and the fact that *nonce* is a randomly chosen value, P_j is able to answer to such a query only if it *possesses* the PUF. The final step to obtain UC security for any functionality consists in using the compiler of [22], which only needs a UC secure implementation of the $\mathcal{F}_{\mathsf{mcom}}$ functionality.

2.3.1 Sub-Protocols

In this section we provide more details for Protocol $\mathsf{Com}_{\mathsf{shext}}$ and Protocol $\mathsf{Com}_{\mathsf{equiv}}$, which are the main ingredients of Protocol $\mathsf{Com}_{\mathsf{uc}}$. For simplicity, in this section we use

[4]Note that however our protocol enjoys parallel composition and reuse of the same PUF, i.e., one can commit to a string reusing the same PUF.

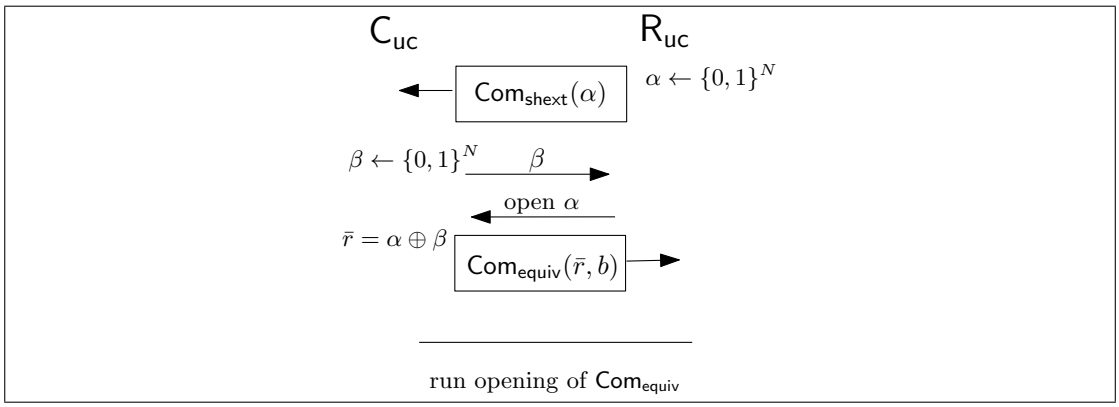

Figure 2.5: *Pictorial representation of Protocol* $\mathsf{Com}_{\mathsf{uc}}$.

Committer's Input: Bit $b \in \{0,1\}$.
Commitment Phase

$\mathsf{R}_{\mathsf{uc}} \Leftrightarrow \mathsf{C}_{\mathsf{uc}}$: (Coin Flipping)

1. R_{uc} picks $\alpha \xleftarrow{\$} \{0,1\}^N$; commit to α by running $((\mathbf{c}_\alpha, d_\alpha), \mathbf{c}_\alpha) \leftarrow \langle \mathsf{C}_{\mathsf{shext}}(\mathtt{com}, \alpha), \mathsf{R}_{\mathsf{shext}}(\mathtt{recv}) \rangle$ with C_{uc}.
2. C_{uc} sends $\beta \xleftarrow{\$} \{0,1\}^N$ to R_{uc}.
3. R_{uc} sends decommitment (α, d_α) to C_{uc}.
4. C_{uc}: if $\mathsf{R}_{\mathsf{shext}}(d_\alpha, \alpha) = 0$, abort.

$\mathsf{C}_{\mathsf{uc}} \Leftrightarrow \mathsf{R}_{\mathsf{uc}}$: (Equivocal Commitment)
C_{uc} commit to b by running $((\mathbf{c}_{\mathsf{bit}}, d_{\mathsf{bit}}), \mathbf{c}_{\mathsf{bit}}) \leftarrow \langle \mathsf{C}_{\mathsf{equiv}}(\mathtt{com}, b), \mathsf{R}_{\mathsf{equiv}}(\mathtt{recv}) \rangle (\alpha \oplus \beta)$ with R_{uc}.

Decommitment Phase

C_{uc} sends decommitment (b, d_{bit}) to R_{uc}.

R_{uc} accepts iff $\mathsf{R}_{\mathsf{equiv}}(\alpha \oplus \beta, \mathbf{c}_{\mathsf{bit}}, b, d_{\mathsf{bit}})$ is accepting. Else, reject.

Figure 2.6: *Computational UC Commitment Scheme* $(\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$.

the following informal notation. We refer to a PUF created by party A as PUF_A, and we denote by $v \leftarrow \mathsf{PUF}_A(q)$ the evaluation of the PUF PUF_A on challenge q.

Statistically Hiding Straight-line Extractable Commitment Scheme. Let $\mathsf{Com}_{\mathsf{SH}} = (\mathsf{C}_{\mathsf{SH}}, \mathsf{R}_{\mathsf{SH}})$ be a Statistically Hiding string commitment scheme, $(\mathsf{S}_{\mathsf{OT}}, \mathsf{S}_{\mathsf{OT}})$ be a statistical receiver OT protocol (namely, an OT protocol where the receiver's privacy is statistically preserved). Let (P, V) be a Statistical Zero Knowledge Argument of Knowledge

($\mathcal{SZK}$AoK) for the following relation: $\mathcal{R}_{\mathsf{com}} = \{(\mathbf{c}, (s, d))$ such that $\mathsf{R}_{\mathsf{SH}}(\mathbf{c}, s, d) = 1\}$. Protocol $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$ is depicted in Fig.2.7.

Committer's Input: Bit $b \in \{0, 1\}$.
Commitment Phase

$\mathsf{R}_{\mathsf{shext}}$: Initialize $\mathsf{PUF}_{\mathsf{R}}$.

1. obtain $a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$, $a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$, for $(q_0, q_1) \xleftarrow{\$} \{0,1\}^n$.
2. $(st_0, p_0) \leftarrow \mathsf{FuzGen}(a_0)$, $(st_1, p_1) \leftarrow \mathsf{FuzGen}(a_1)$.
3. handover $\mathsf{PUF}_{\mathsf{R}}$ to $\mathsf{C}_{\mathsf{shext}}$.

$\mathsf{R}_{\mathsf{shext}} \Leftrightarrow \mathsf{C}_{\mathsf{shext}}$: (Statistical OT phase)
$\langle \mathsf{S}_{\mathsf{OT}}(q_0, q_1), \mathsf{S}_{\mathsf{OT}}(b) \rangle$ is run by $\mathsf{R}_{\mathsf{shext}}$ as S_{OT} with input (q_0, q_1), and $\mathsf{C}_{\mathsf{shext}}$ as S_{OT} with input b. Let q'_b be the local output of $\mathsf{C}_{\mathsf{shext}}$.

$\mathsf{C}_{\mathsf{shext}}$: $a'_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q'_b)$. If $\mathsf{PUF}_{\mathsf{R}}$ aborts, $a'_b \xleftarrow{\$} \{0,1\}^l$.

$\mathsf{C}_{\mathsf{shext}} \Leftrightarrow \mathsf{R}_{\mathsf{shext}}$: (Statistically Hiding Commitment)
$((\mathbf{c}, d), \mathbf{c}) \leftarrow \langle \mathsf{C}_{\mathsf{SH}}(\mathtt{com}, a'_b), \mathsf{R}_{\mathsf{SH}}(\mathtt{recv}) \rangle$ is run by $\mathsf{C}_{\mathsf{shext}}$ as C_{SH} to commit to a'_b, and by $\mathsf{R}_{\mathsf{shext}}$ as R_{SH}.

$\mathsf{C}_{\mathsf{shext}} \Leftrightarrow \mathsf{R}_{\mathsf{shext}}$: ($\mathcal{SZK}$AoK)
$\langle P(d, a'_b), V \rangle(\mathbf{c})$ is run by $\mathsf{C}_{\mathsf{shext}}$ playing as prover P for the theorem $(\mathbf{c}, (\mathbf{c}, d)) \in \mathcal{R}_{\mathsf{com}}$ and by $\mathsf{R}_{\mathsf{shext}}$ playing as verifier V on input $\mathbf{c}$. If the proof is not accepting, $\mathsf{R}_{\mathsf{shext}}$ aborts.

Decommitment Phase

$\mathsf{C}_{\mathsf{shext}}$: if $\mathsf{PUF}_{\mathsf{R}}$ did not abort, send opening (d, a'_b, b) to $\mathsf{R}_{\mathsf{shext}}$.

$\mathsf{R}_{\mathsf{shext}}$: if $\mathsf{R}_{\mathsf{SH}}(\mathbf{c}, a'_b, d) = 1$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b$ then accept. Else reject.

Figure 2.7: *Statistically Hiding Straight-Line Extractable Bit Commitment Scheme* $(\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$.

Theorem 2. *If* $\mathsf{Com}_{\mathsf{SH}} = (\mathsf{C}_{\mathsf{SH}}, \mathsf{R}_{\mathsf{SH}})$ *is a statistically-hiding commitment scheme,* $(\mathsf{S}_{\mathsf{OT}}, \mathsf{S}_{\mathsf{OT}})$ *is a statistical receiver OT protocol and* (P, V) *is a* $\mathcal{SZK}$AoK*, then* $\mathsf{Com}_{\mathsf{shext}}$ *is a statistically hiding straight-line extractable bit commitment scheme in the malicious PUFs model.*

Proof. **Completeness.** Before delivering its own PUF $\mathsf{PUF}_{\mathsf{R}}$, $\mathsf{R}_{\mathsf{shext}}$ queries it with a pair of random challenges (q_0, q_1) and gets answers (a_0, a_1). To commit to a bit b, $\mathsf{C}_{\mathsf{shext}}$ has to commit to the output a_b of $\mathsf{PUF}_{\mathsf{R}}$.

By the completeness of the OT protocol, $\mathsf{C}_{\mathsf{shext}}$ obtains the query q_b corresponding to its secret bit. Then $\mathsf{C}_{\mathsf{shext}}$ queries $\mathsf{PUF}_{\mathsf{R}}$ with q_b and commits to the response a'_b run-

ning $\mathsf{C_{SH}}$. Furthermore, $\mathsf{C_{shext}}$ proves using $\mathcal{SZK}\mathsf{AoK}$ the knowledge of the opening. By the completeness of $\mathcal{SZK}\mathsf{AoK}$ and $\mathsf{Com_{SH}}$ the commitment phase is concluded without aborts. In the opening phase, $\mathsf{C_{shext}}$ sends b and opens the commitment to a'_b, and $\mathsf{R_{shext}}$ checks whether the string a'_b matches the answer a_b obtained by its own PUF applying the fuzzy extractor. By the response consistency property, $\mathsf{R_{shext}}$ gets the correct answer and accept the decommitment for the bit b.

Statistically Hiding. We show that, for all $\mathsf{R}^*_{\mathsf{shext}}$ it holds that:

$$\mathsf{view}_{\mathsf{R}^*_{\mathsf{shext}}}(\mathsf{C_{shext}}(\mathtt{com}, 0), \mathsf{R_{shext}}) \stackrel{\mathsf{S}}{\equiv} \mathsf{view}_{\mathsf{R}^*_{\mathsf{shext}}}(\mathsf{C_{shext}}(\mathtt{com}, 1), \mathsf{R}^*_{\mathsf{shext}}).$$

This follows from the statistical security of the three sub-protocols run in the commitment phase by $\mathsf{C_{shext}}$. More specifically, recall that the view of $\mathsf{R}^*_{\mathsf{shext}}$ in the commitment phase consists of the transcript of the execution of the OT protocol $(\mathsf{S_{OT}}, \mathsf{S_{OT}})$, the transcript of the Statistically Hiding commitment scheme $\mathsf{Com_{SH}}$ and the transcript of the execution of the $\mathcal{SZK}\mathsf{AoK}$ protocol. The proof goes by hybrids.

H_0**:** In this hybrid the sender $\mathsf{C_{shext}}$ commits to bit 0. Namely, it plays the OT protocol with the bit 0 to obtain q'_0, then it queries the malicious $\mathsf{PUF}^*_{\mathsf{R}}$ to obtain a string a'_0, then it commits to a'_0 executing $\mathsf{C_{SH}}$ and finally it runs the honest prover P to prove knowledge of the decommitment.

H_1**:** In this hybrid, $\mathsf{C_{shext}}$ proceeds as in H_0, except that it executes the zero knowledge protocol by running the zero knowledge simulator S. By the statistical zero knowledge property of (P, V), hybrids H_0 and H_1 are statistically indistinguishable.

H_2**:** In this hybrid, $\mathsf{C_{shext}}$ proceeds as in H_1, excepts that it runs $\mathsf{C_{SH}}$ to commit to a random string s instead of a'_0. By the statistically hiding property of protocol $\mathsf{Com_{SH}}$, hybrids H_1 and H_2 are statistically indistinguishable.

H_3**:** In this hybrid, $\mathsf{C_{shext}}$ proceeds as in H_2, except that in OT protocol it plays with bit 1, obtaining query q'_1. By the receiver security of protocol $(\mathsf{S_{OT}}, \mathsf{S_{OT}})$, hybrids H_2 and H_3 are statistically indistinguishable.

H_4**:** In this hybrid, $\mathsf{C_{shext}}$ proceeds as in H_3, except that here it queries the PUF with string q'_1 to obtain a'_1 (however it still commits to the random string s). If the $\mathsf{PUF}^*_{\mathsf{R}}$ aborts, then $\mathsf{C_{shext}}$ sets $a'_1 \leftarrow \{0,1\}^l$. Note that any malicious behavior does not effect the transcript generated in H_4. Thus, hybrids H_3 and H_3 are identical.

H_5**:** In this hybrid, $\mathsf{C_{shext}}$ proceeds as in H_4 except that it commits to the string a'_1. By the statistically hiding property of protocol $\mathsf{Com_{SH}}$, hybrids H_4 and H_5 are statistically indistinguishable.

H_6**:** In this hybrid, $\mathsf{C_{shext}}$ proceeds as in H_5, except that it executes the zero knowledge protocol running as the honest prover P. By the statistical zero knowledge property of (P, V), hybrids H_5 and H_6 are statistically indistinguishable.

By observing that hybrid H_0 corresponds to the case in which $\mathsf{C}_{\mathsf{shext}}$ commits to 0 and hybrid H_6 corresponds to the case in which $\mathsf{C}_{\mathsf{shext}}$ commits to 1, the hiding property is proved.

Straight-line Extractability. To prove extractability we show a straight-line strict polynomial-time extractor E that satisfies the properties required by Definition 18. Recall that, in the commitment scheme $\mathsf{Com}_{\mathsf{shext}}$, the sender basically commits to the answer a_b received from $\mathsf{PUF}_{\mathsf{R}}$. By the unpredictability of PUF, the sender needs to get the right query q_b from $\mathsf{R}_{\mathsf{shext}}$ in order to obtain the value to commit to. Such q_b is obliviously retrieved by $\mathsf{C}_{\mathsf{shext}}$ running OT with the bit b. The strategy of the extractor, that we show below, is very simple. It consists of running the commitment phase as the honest receiver, and then looking at the queries made by $\mathsf{C}^*_{\mathsf{shext}}$ to $\mathsf{PUF}_{\mathsf{R}}$ to detect which among q_0, q_1 has been asked and thus extract the bit. The extraction of the bit fails when one of the following two cases happens. Case **Fail1**: the set of queries contains both (q_0, q_1) (or at least a pair that is within their hamming distance); in this case E cannot tell which is the bit played by $\mathsf{C}^*_{\mathsf{shext}}$ and therefore outputs $\perp$. By the sender's security of OT this case happens only with negligible probability. Case **Fail2**: the set of queries does not contain any query close (within hamming distance) to neither q_0 nor q_1. This is also a bad case since E cannot extract any information. However, if there exists such a $\mathsf{C}^*_{\mathsf{shext}}$ that produces an accepting commitment without querying the PUF in the commitment phase (but perhaps it makes queries in the decommitment phase only) then, given that responses of honest PUFs are unpredictable, one can break either the binding property of the underlying commitment scheme $\mathsf{Com}_{\mathsf{SH}}$ or the argument of knowledge property of (P, V). The formal description of E is given below. Formal arguments follow.

Extractor E

Commitment Phase. Run the commitment phase following the honest receiver procedure. We denote by (q_0, q_1) the queries made by the extractor E to the honest PUF before delivering it to $\mathsf{C}^*_{\mathsf{shext}}$. E uses such a pair when running as S_{OT} in OT protocol. If all sub-protocols (OT, $\mathsf{Com}_{\mathsf{SH}}, \mathcal{SZK}\mathsf{AoK}$) are successfully completed go the extraction phase. Else, abort.

Extraction phase. Let $\mathcal{Q}$ be the set of queries asked by $\mathsf{C}^*_{\mathsf{shext}}$ to $\mathsf{PUF}_{\mathsf{R}}$ during the commitment phase.

- **Fail1.** If there exists a pair $q'_0, q'_1 \in \mathcal{Q}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q'_0) \leq d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q'_1) \leq d_{\mathsf{min}}$, output $b^\star = \perp$.
- **Fail2.** If for all $q' \in \mathcal{Q}$ it holds that $\mathsf{dis}_{\mathsf{ham}}(q_0, q') > d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q') > d_{\mathsf{min}}$, output $b^\star = \perp$.
- **Good.** 1. If there exists $q' \in \mathcal{Q}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q') \leq d_{\mathsf{min}}$ then output $b^\star = 0$.
  2. If there exists $q' \in \mathcal{Q}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_1, q') \leq d_{\mathsf{min}}$ then output $b^\star = 1$.

The above extractor E satisfies the following three properties.

Simulation. E follows the procedure of the honest receiver $\mathsf{R}_{\mathsf{shext}}$. Thus the view of $\mathsf{C}^*_{\mathsf{shext}}$ playing with E is identical to the view of $\mathsf{C}^*_{\mathsf{shext}}$ playing with $\mathsf{R}_{\mathsf{shext}}$.

Extraction. Let τ_c the transcript of the commitment phase. For the extraction property we have to show that if τ_c is accepting, then the probability that E outputs $\perp$ is negligible. Note that E outputs $\perp$ if and only if one of the event between **Fail1** and **Fail2** happens. Thus,

$$\mathsf{Pr}\,[\,b^\star = \perp\,] = \mathsf{Pr}\,[\,\mathbf{Fail1}\,] + \mathsf{Pr}\,[\,\mathbf{Fail2}\,]$$

In the following we show that, if τ_c is accepting, then $\mathsf{Pr}\,[\,b^\star = \perp\,]$ is negligible by showing separately that $\mathsf{Pr}\,[\,\mathbf{Fail1}\,]$ and $\mathsf{Pr}\,[\,\mathbf{Fail2}\,]$ are negligible.

Lemma 1 ($\mathsf{Pr}\,[\,\mathbf{Fail1}\,]$ is negligible)**.** *If* $(\mathsf{S_{OT}}, \mathsf{S_{OT}})$ *is an Oblivious Transfer protocol, then* $\mathsf{Pr}\,[\,\mathbf{Fail1}\,]$ *is negligible.*

Proof. Assume that there exists a PPT $\mathsf{C}^*_{\mathsf{shext}}$ such that event **Fail1** happens with non-negligible probability δ. Then it is possible to construct $\mathsf{R}^*_{\mathsf{OT}}$ that uses $\mathsf{C}^*_{\mathsf{shext}}$ to break the sender's security of the OT protocol. $\mathsf{R}^*_{\mathsf{OT}}$ interacts with an external OT sender $\mathsf{S_{OT}}$, on input auxiliary information $z = (s_0, s_1)$, while it runs $\mathsf{C}^*_{\mathsf{shext}}$ internally. $\mathsf{R}^*_{\mathsf{OT}}$ initializes and sends $\mathsf{PUF_R}$ to $\mathsf{C}^*_{\mathsf{shext}}$, then it runs the OT protocol forwarding the messages received from the external sender $\mathsf{S_{OT}}$ to $\mathsf{C}^*_{\mathsf{shext}}$ and vice versa. When the OT protocol is completed, $\mathsf{R}^*_{\mathsf{OT}}$ continues the internal execution with $\mathsf{C}^*_{\mathsf{shext}}$ emulating the honest receiver. When the commitment phase is successfully completed, $\mathsf{R}^*_{\mathsf{OT}}$ analyses the set $\mathcal{Q}$ of queries made by $\mathsf{C}^*_{\mathsf{shext}}$ to $\mathsf{PUF_R}$. If there exists a pair (q'_0, q'_1) within hamming distance with strings (s_0, s_1) then $\mathsf{R}^*_{\mathsf{OT}}$ outputs (s_0, s_1), therefore breaking the sender's security of OT with probability δ (indeed, there exists no simulator that can simulate such attack since in the ideal world Sim gets only one input among (s_0, s_1)). Since by assumption $(\mathsf{S_{OT}}, \mathsf{S_{OT}})$ is a stand-alone secure OT protocol, δ must be negligible. □

Lemma 2 ($\mathsf{Pr}\,[\,\mathbf{Fail2}\,]$ is negligible)**.** *Assume that* τ_c *is an accepting transcript. If* $\mathsf{Com_{SH}} = (\mathsf{C_{SH}}, \mathsf{R_{SH}})$ *is a commitment scheme and if* (P, V) *is a* $\mathcal{SZK}\mathsf{AoK}$ *then* $\mathsf{Pr}\,[\,\mathbf{Fail2}\,]$ *is negligible.*

Proof. If transcript τ_c is accepting then it holds that $\mathsf{C}^*_{\mathsf{shext}}$ in the decommitment phase will send a tuple (b, d, a'_b) for which, given τ_c, the receiver $\mathsf{R_{shext}}$ accepts, i.e., the opening (d) of the statistically hiding commitment is valid *and* corresponds to an answer (a'_b) of $\mathsf{PUF_R}$ upon one of the queries played by the $\mathsf{R_{shext}}$ in the OT protocol. Formally, $\mathsf{R_{SH}}(\mathbf{c}, a'_b, d) = 1$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b$.

Toward a contradiction, assume that $\mathsf{Pr}\,[\,\mathbf{Fail2}\,] = \delta$ and is not-negligible. Recall that the event **Fail2** happens when $\mathsf{C}^*_{\mathsf{shext}}$ successfully completed the commitment phase, without querying $\mathsf{PUF_R}$ with any of (q_0, q_1). Given that τ_c is accepting, let (b, d, a'_b) be an accepting decommitment, we have the following cases:

1. $\mathsf{C}^*_{\mathsf{shext}}$ honestly committed to the correct a'_b without having queried $\mathsf{PUF_R}$. By the unpredictability of $\mathsf{PUF_R}$ we have that this case has negligible probability to happen.
2. $\mathsf{C}^*_{\mathsf{shext}}$ queries $\mathsf{PUF_R}$ in the *decommitment* phase to obtain the value a'_b to be opened. Thus $\mathsf{C}^*_{\mathsf{shext}}$ opens commitment $\mathbf{c}$ (sent in the commitment phase) as string a'_b.

We argue that by the computational binding of $\mathsf{Com}_{\mathsf{SH}}$ and by the argument of knowledge property of (P, V) this case also happens with negligible probability.
First, we show and adversary $\mathsf{C}^*_{\mathsf{SH}}$ that uses $\mathsf{C}^*_{\mathsf{shext}}$ as a black-box to break the binding of the commitment scheme $\mathsf{Com}_{\mathsf{SH}}$ with probability δ. $\mathsf{C}^*_{\mathsf{SH}}$ runs $\mathsf{C}^*_{\mathsf{shext}}$ internally, simulating the honest receiver $\mathsf{R}_{\mathsf{shext}}$ to it, and forwarding only the messages belonging to $\mathsf{Com}_{\mathsf{SH}}$ to an external receiver R_{SH}, and vice versa. Let $\mathbf{c}$ denote the transcript of $\mathsf{Com}_{\mathsf{SH}}$. When the commitment phase of $\mathsf{Com}_{\mathsf{shext}}$ is successfully completed, and therefore $\mathsf{C}^*_{\mathsf{shext}}$ has provided an accepting proof for the theorem $(\mathbf{c}, \cdot) \in \mathcal{R}_{\mathtt{com}}$, $\mathsf{C}^*_{\mathsf{SH}}$ runs the extractor E_P associated to the protocol (P, V). By the argument of knowledge property, E_P, having oracle access to $\mathsf{C}^*_{\mathsf{shext}}$, extracts the witness $(\tilde{a}_b, \tilde{d})$ used by $\mathsf{C}^*_{\mathsf{shext}}$ to prove theorem $\mathbf{c} \in \mathcal{R}_{\mathtt{com}}$ w.h.p. If the witness extracted is not a valid decommitment of $\mathbf{c}$, then $\mathsf{C}^*_{\mathsf{shext}}$ can be used to break the soundness of (P, V).
Else, $\mathsf{C}^*_{\mathsf{SH}}$ proceeds to the decommitment phase, and as by hypothesis of Lemma 2, since the commitment τ_c is accepting, $\mathsf{C}^*_{\mathsf{shext}}$ provides a valid opening (a_b, d).
If $(\tilde{a}_b, \tilde{d}) \neq (a_b, d)$ are two valid openings for $\mathbf{c}$ then $\mathsf{C}^*_{\mathsf{SH}}$ outputs such tuple breaking the binding property of $\mathsf{Com}_{\mathsf{SH}}$ with probability δ.
If $(\tilde{a}_b, \tilde{d}) = (a_b, d)$ with non-negligible probability, then consider the following analysis. By assumption, event **Fail2** happens when $\mathsf{C}^*_{\mathsf{shext}}$ does not query $\mathsf{PUF}_{\mathsf{R}}$ with none among (q_0, q_1). By the unpredictability property, it holds that without querying the PUF, $\mathsf{C}^*_{\mathsf{shext}}$ cannot guess the values a_b, thus w.h.p. the commitment $\mathbf{c}$ played by $\mathsf{C}^*_{\mathsf{shext}}$ in the commitment phase, does not hide the value a_b. However, since the output of the extraction is a valid opening for a_b, then it must have been the case that in one of the rewinding attempts of the black-box extractor E_P, $\mathsf{C}^*_{\mathsf{shext}}$ has obtained a_b by asking $\mathsf{PUF}_{\mathsf{R}}$. Indeed, upon each rewind E_P very luckily changes the messages played by the verifier of the ZK protocol, and $\mathsf{C}^*_{\mathsf{shext}}$ could choose the queries for $\mathsf{PUF}_{\mathsf{R}}$ adaptively on such messages. However, recalling that E_P is run by $\mathsf{C}^*_{\mathsf{SH}}$ to extract from $\mathsf{C}^*_{\mathsf{shext}}$, $\mathsf{C}^*_{\mathsf{SH}}$ can avoid such failure by following this strategy: when a rewinding thread leads $\mathsf{C}^*_{\mathsf{shext}}$ to ask the PUF with query q_b, then abort such thread and start a new one. By noticing that in the commitment phase, $\mathsf{C}^*_{\mathsf{shext}}$ did not query the PUF with q_b, we have that, by the argument of knowledge property of (P, V) this event happens again in the rewinding threads w.h.p. Thus, by discarding the rewinding thread in which $\mathsf{C}^*_{\mathsf{shext}}$ asks for query q_b, $\mathsf{C}^*_{\mathsf{SH}}$ is still be able to extract the witness in polynomial time (again, if this was not the case then one can use $\mathsf{C}^*_{\mathsf{shext}}$ to break the argument of knowledge property). With this strategy, the event $(\tilde{a}_b, \tilde{d}) = (a_b, d)$ is ruled out.

□

Binding. Let $b^\star = b_0$ the bit extracted by E, given the transcript τ_c. Assume that in the decommitment phase $\mathsf{C}^*_{\mathsf{shext}}$ provides a valid opening of τ_c as b_1 and $b_0 \neq b_1$. If such an event happens, the the following three events happened: 1) in the commitment phase $\mathsf{C}^*_{\mathsf{shext}}$ queried $\mathsf{PUF}_{\mathsf{R}}$ with query q_{b_0} only; 2) in decommitment phase $\mathsf{C}^*_{\mathsf{shext}}$ queried $\mathsf{PUF}_{\mathsf{R}}$ with q_{b_1}, let a_{b_1} be the answer; 3)$\mathsf{C}^*_{\mathsf{shext}}$ opens the commitment $\mathbf{c}$ (that is the commitment of the answer of $\mathsf{PUF}_{\mathsf{R}}$ received in the commitment phase), as a_{b_1}, but $\mathbf{c}$ was computed

without knowledge of $\mathsf{PUF}_{\mathsf{R}}(q_{b_1})$.

By the security of the OT protocol and by the computational binding of the commitment scheme $\mathsf{Com}_{\mathsf{SH}}$, the above cases happen with negligible probability. Formal arguments follow previous discussions and are therefore omitted. □

Lemma 3. *Protocol* $\mathsf{Com}_{\mathsf{shext}}$ *is close under parallel repetition using the same PUF.*

Sketch. The proof comes straightforwardly by the fact that all sub-protocols used in protocol $\mathsf{Com}_{\mathsf{shext}}$ are close under parallel repetition. However, issues can arise when the same, possibly malicious and stateful PUF, is reused. Note that, the output of the (malicious) PUF is statistically hidden in the commitment phase and that it is revealed only in the decommitment phase. Thus, any side information that is leaked by a dishonest PUF, cannot be used by the malicious creator, before the decommitment phase. At the decommitment stage however, the input of the committer is already revealed, and no more information is therefore gained by the malicious party. We stress out that re-usability is possible only when many instances of $\mathsf{Com}_{\mathsf{shext}}$ are run in *parallel*, i.e., only when all decommitment happen simultaneously. If decommitment phases are interleaved with commitment phase of other sessions, then reusing the same PUF, allow the malicious creator to gain information about sessions that are not open yet. To see why, let i and j be two concurrent executions. Assume that the commitment of i and j is done in parallel but session j is decommitted before session i. Then, a malicious PUF can send information on the bit committed in the session i through the string sent back for the decommitment of j. □

Statistically Hiding Straight-line Extractable String Commitment Scheme. We obtain statistically hiding straight-line extractable *string* commitment scheme, for n-bit string, by running n execution of $\mathsf{Com}_{\mathsf{shext}}$ in parallel and reusing the same PUF. In the main protocol shown in Figure 5.3 we use the same notation $\mathsf{Com}_{\mathsf{shext}}$ to refer to a string commitment scheme.

Statistically Binding Straight-line Extractable/Equivocal Commitment Scheme. Let $l = rg(n)$ be the range of the PUF, $(\mathsf{S}_{\mathsf{OT}}, \mathsf{S}_{\mathsf{OT}})$ be a statistical receiver OT protocol and let $G : \{0,1\}^n \to \{0,1\}^{3n}$ be a PRG. The commitment scheme that we present, takes as common input a string $\bar{r} = r_1, \ldots, r_l$, that is *uniformly chosen* in the set $(\{0,1\}^{3n})^l$. This string can be seen as l distinct parameters for Naor's commitment, and indeed it is used to commit bit-by-bit to an l-bit string (i.e., the answer received from the PUF). Our statistically binding straight-line extractable and equivocal commitment scheme $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$ is depicted in Fig. 2.8.

Theorem 3. *If G is a PRG and $(\mathsf{S}_{\mathsf{OT}}, \mathsf{S}_{\mathsf{OT}})$ is statistical receiver OT protocol, then $\mathsf{Com}_{\mathsf{equiv}} = (\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$ is a statistically binding straight-line extractable and equivocal commitment scheme in the malicious PUFs model.*

Proof. **Completeness.** It follows from the completeness of the OT protocol, the correctness of Naor's commitment and the response consistency property of PUFs with

Committer's Input: Bit $b \in \{0,1\}$. **Common Input:** $\bar{r} = (r_1, \ldots, r_l)$

Commitment Phase

$\mathsf{R}_{\mathsf{equiv}}$: Initialize $\mathsf{PUF}_{\mathsf{R}}$;

1. obtain $a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$, $a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$, for $(q_0, q_1) \xleftarrow{\$} \{0,1\}^n$.
2. $(st_0, p_0) \leftarrow \mathsf{FuzGen}(a_0)$, $(st_1, p_1) \leftarrow \mathsf{FuzGen}(a_1)$.
3. handover $\mathsf{PUF}_{\mathsf{R}}$ to $\mathsf{C}_{\mathsf{equiv}}$;
4. choose random tape $\mathsf{ran}_{\mathsf{OT}} \xleftarrow{\$} \{0,1\}^*$.

$\mathsf{R}_{\mathsf{equiv}} \Leftrightarrow \mathsf{C}_{\mathsf{equiv}}$: (OT phase)
$\langle \mathsf{S}_{\mathsf{OT}}(q_0, q_1), \mathsf{S}_{\mathsf{OT}}(b) \rangle$ is run by $\mathsf{R}_{\mathsf{equiv}}$ as S_{OT} with input (q_0, q_1) and randomness $\mathsf{ran}_{\mathsf{OT}}$, while $\mathsf{C}_{\mathsf{equiv}}$ runs as S_{OT} with input b. Let q'_b be the local output of $\mathsf{C}_{\mathsf{equiv}}$, and τ_{OT} be the transcript of the execution of the OT protocol.

$\mathsf{C}_{\mathsf{equiv}}$:(Statistically Binding Commitment)

1. $a'_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q'_b)$. If $\mathsf{PUF}_{\mathsf{R}}$ aborts, $a'_b \xleftarrow{\$} \{0,1\}^l$.
2. for $1 \leq i \leq l$, pick $s_i \xleftarrow{\$} \{0,1\}^n$, $\mathbf{c}_i = G(s_i) \oplus (r_i \wedge a'_b[i])$. [a]
3. send $\mathbf{c}_1, \ldots, \mathbf{c}_l$ to $\mathsf{R}_{\mathsf{equiv}}$.

$\mathsf{R}_{\mathsf{equiv}}$: upon receiving $\mathbf{c}_1, \ldots, \mathbf{c}_l$, send $\mathsf{ran}_{\mathsf{OT}}, q_0, q_1$ to $\mathsf{C}_{\mathsf{equiv}}$.

$\mathsf{C}_{\mathsf{equiv}}$: check if transcript τ_{OT} is consistent with $(\mathsf{ran}_{\mathsf{OT}}, q_0, q_1, b)$. If the check fails abort.

Decommitment Phase

$\mathsf{C}_{\mathsf{equiv}}$: if $\mathsf{PUF}_{\mathsf{R}}$ did not abort, send $((s_1, \ldots, s_l), a'_b), b$ to $\mathsf{R}_{\mathsf{shext}}$.

$\mathsf{R}_{\mathsf{equiv}}$: if for all i, it holds that ($\mathbf{c}_i = G(s_i) \oplus (r_i \wedge a'_b[i])$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b$) then accept. Else reject.

[a] where $(r_i \wedge a'_b[i])_j = r_i[j] \wedge a'_b[i]$.

Figure 2.8: *Statistically Binding Straight-line Extractable and Equivocal Commitment* $(\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$.

fuzzy extractors. To commit to the bit b, the sender $\mathsf{C}_{\mathsf{equiv}}$ is required to commit to the answer of $\mathsf{PUF}_{\mathsf{R}}$ on input q_b. Therefore, $\mathsf{C}_{\mathsf{equiv}}$ runs the OT protocol with input b and obtains the query q_b and thus the value to commit to using Naor's commitments. The correctness of OT guarantees that the consistency check performed by $\mathsf{C}_{\mathsf{equiv}}$ goes through. In the decommitment phase, the response consistency property along with correctness of Naor, allow the receiver $\mathsf{R}_{\mathsf{equiv}}$ to obtain the string a_b and in therefore the bit decommitted to by $\mathsf{C}_{\mathsf{equiv}}$.

Straight-line Extractability.
Extractor E

Commitment Phase. Run the commitment phase following the honest receiver procedure: E queries $\mathsf{PUF_R}$ with (q_0, q_1) before delivering it to $\mathsf{C}^*_{\mathsf{equiv}}$, and uses such a pair when running as $\mathsf{S_{OT}}$ in OT protocol. If OT protocol is not successfully completed then abort. Else, let $\mathcal{Q}_{\mathsf{precom}}$ be the set of queries asked by $\mathsf{C}^*_{\mathsf{equiv}}$ to $\mathsf{PUF_R}$ *before* sending the commitments $\mathbf{c}_1, \ldots, \mathbf{c}_l$ to E. Upon receiving such commitments, do as follows:

Fail1. If there exists a pair $q'_0, q'_1 \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis_{ham}}(q_0, q'_0) \leq d_{\mathsf{min}}$ and $\mathsf{dis_{ham}}(q_1, q'_1) \leq d_{\mathsf{min}}$, output $b^\star = \perp$.

Fail2. If for all $q' \in \mathcal{Q}_{\mathsf{precom}}$ it holds that $\mathsf{dis_{ham}}(q_0, q') > d_{\mathsf{min}}$ and $\mathsf{dis_{ham}}(q_1, q') > d_{\mathsf{min}}$, output $b^\star = \perp$.

Good. 1. If there exists $q' \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis_{ham}}(q_0, q') \leq d_{\mathsf{min}}$ then output $b^\star = 0$;
2. If there exists $q' \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis_{ham}}(q_1, q') \leq d_{\mathsf{min}}$ then output $b^\star = 1$;

Finally sends $\mathsf{ran_{OT}}, q_0, q_1$ to $\mathsf{C}^*_{\mathsf{equiv}}$.
Simulation. E follows the procedure of the honest receiver $\mathsf{R}_{\mathsf{equiv}}$. Thus the view of $\mathsf{C}^*_{\mathsf{equiv}}$ playing with E is identical to the view of $\mathsf{C}^*_{\mathsf{equiv}}$ playing with $\mathsf{R}_{\mathsf{equiv}}$.
Extraction. The proof of extraction follows from the same arguments shown in the proof of Theorem 2, and it is simpler since in protocol $\mathsf{Com}_{\mathsf{equiv}}$ we use statistically binding commitments (given that the common parameter $\bar{r}$ is uniformly chosen).

Let τ_c the transcript of the commitment phase. For the extraction property we have to show that if τ_c is accepting, then the probability that E outputs $\perp$ is negligible. Note that E outputs $\perp$ if and only if one event between **Fail1** and **Fail2** happens. Thus,

$$\mathsf{Pr}\left[\, b^\star = \perp \,\right] = \mathsf{Pr}\left[\, \mathbf{Fail1} \,\right] + \mathsf{Pr}\left[\, \mathbf{Fail2} \,\right]$$

By the sender's security property of the OT protocol, event **Fail1** happens with negligible probability. The formal proof follows the same arguments given in Lemma 1. Given that the common parameter $\bar{r}$ is uniformly chosen, we have that the Naor's commitments (i.e., $\mathbf{c}_1, \ldots, \mathbf{c}_l$) sent by $\mathsf{C}^*_{\mathsf{equiv}}$ in the commitment phase, are statistically binding. Thus, by the unpredictability property of PUFs and the by the statistically binding property of Naor's commitment scheme, event **Fail2** also happens with negligible probability only.
Binding. Given that the common input $\bar{r}$ is uniformly chosen, binding of $\mathsf{Com}_{\mathsf{equiv}}$ follows from the statistically binding property of Naor's commitment scheme.
Straight-line Equivocality. In the following we show a straight-line simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ and we prove that the view generated by the interaction between $\mathcal{S}$ and $\mathsf{R}^*_{\mathsf{equiv}}$ is computationally indistinguishable from the view generated by the interaction between $\mathsf{C}_{\mathsf{equiv}}$ and $\mathsf{R}^*_{\mathsf{equiv}}$.

$\underline{\mathcal{S}_1}$. $(\bar{r} = r_1, \ldots, r_l, \xi_1) \leftarrow \mathcal{S}_1(1^{ln})$:
For $i = 1, \ldots, l$.

1. pick $s_0^i \leftarrow \{0,1\}^n$, $\alpha_0^i \leftarrow G(s_0^i)$;
2. pick $s_1^i \leftarrow \{0,1\}^n$, $\alpha_1^i \leftarrow G(s_1^i)$;
3. $r_i = \alpha_0^i \oplus \alpha_1^i$.

Output $r_1, \ldots, r_l$, $\xi_1 = \{s_0^i, s_1^i\}_{i \in l}$;

$\underline{\mathcal{S}_2}$. $(\xi_2) \leftarrow \mathcal{S}_2(\xi_1)$:

- obtain $\mathsf{PUF}_{\mathsf{R}}^*$ from $\mathsf{R}_{\mathsf{equiv}}^*$.
- run OT protocol with input a random bit $\tilde{b}$; if the OT protocol is not successfully completed, abort.
- computes commitments as follows: for $i = 1, \ldots, l$, $\tilde{\mathbf{c}}_i \leftarrow G(s_0^i)$. Send $\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_l$ to $\mathsf{R}_{\mathsf{equiv}}^*$.
- Obtain $(\mathsf{ran}_{\mathsf{OT}}, q_0', q_1')$ from $\mathsf{R}_{\mathsf{equiv}}^*$ and check if the transcript τ_{OT} is consistent with it. If the check fails, abort. Else, output $\xi_2 = \{\xi_1, (q_0', q_1')\}$.

$\underline{\mathcal{S}_3}$. $\mathcal{S}_3(\xi_2, b)$:

- query $\mathsf{PUF}_{\mathsf{R}}^*$ with input q_b'. If $\mathsf{PUF}_{\mathsf{R}}^*$ aborts, abort. Otherwise, let a_b' denote the answer of $\mathsf{PUF}_{\mathsf{R}}^*$.
- for $i = 1, \ldots, l$: send $(s^i_{a_b[i]}, a_b[i])$ to $\mathsf{R}_{\mathsf{equiv}}^*$.

Lemma 4. *If $(\mathsf{S}_{\mathsf{OT}}, \mathsf{S}_{\mathsf{OT}})$ is a statistical receiver OT protocol and G is a pseudo-random generator, then for all PPT $\mathsf{R}_{\mathsf{equiv}}^*$ it holds that, $\{\mathsf{out}(\mathbf{Exp}^{\mathsf{C}_{\mathsf{equiv}}}_{\mathsf{R}_{\mathsf{equiv}}^*}(n))\} \stackrel{\mathsf{C}}{\equiv} \mathsf{out}\{(\mathbf{Exp}^{\mathcal{S}}_{\mathsf{R}_{\mathsf{equiv}}^*}(n)\}$.*

Proof. The proof goes by hybrids arguments.

H_0. This is the real world experiment $\mathbf{Exp}^{\mathsf{C}_{\mathsf{equiv}}}_{\mathsf{R}_{\mathsf{equiv}}^*}$.

H_1. In this hybrid the common parameter $\bar{r}$ is chosen running algorithm $\mathcal{S}_1$. The only difference between experiment H_0 and H_1 is in the fact that in H_1 each string $r_i \in \bar{r}$ is pseudo-random. By the pseudo-randomness of PRG H_0 and H_1 are computationally indistinguishable.

H_2. In this hybrid, the commitments $\mathbf{c}_1, \ldots, \mathbf{c}_l$ are computed as in $\mathcal{S}_2$, that is, for all i, $\mathbf{c}_i$ corresponds to an evaluation of the PRG i.e., $\mathbf{c}_i = G(s_0^i)$, regardless of the bit that is committed. Then in the decommitment phase the sender uses knowledge of s_1^i, in case the i-th commitment of a_b' is the bit 1. (Each pair (s_0^i, s_1^i) is inherited from the output of $\mathcal{S}_1$). The difference between experiment H_1 and experiment H_2 is in the fact that in H_2 all commitments are pseudo-random, while in H_1, pseudo-random values are used only to commit to bit 0. By the pseudo-randomness of PRG, experiments H_1 and H_2 are computationally indistinguishable. Note that in this experiment, the sender is not actually committing to the output obtained by querying $\mathsf{PUF}_{\mathsf{R}}^*$.

H_3. In this experiment the sender queries $\mathsf{PUF}^*_{\mathsf{R}}$ on input q_b only in the decommitment phase. The only difference between this experiment and the previous one is that in H_3, the sender is able to detect if $\mathsf{PUF}^*_{\mathsf{R}}$ aborts, only in the decommitment phase. However, in experiment H_2, if the PUF aborts, the sender continues the execution of the commitment phase, committing to a random string, ad aborts only in the decommitment phase. Therefore, hybrids H_2 and H_3 are identical.

H_4. In this experiment, the sender executes the OT protocol with a random bit $\tilde{b}$, obtaining $q_{\tilde{b}}$, but it does not use such a query to evaluate $\mathsf{PUF}^*_{\mathsf{R}}$. Instead it uses the string q'_b received from $\mathsf{R}^*_{\mathsf{equiv}}$ in the last round of the commitment phase.

We stress out that, due to the correctness of the OT protocol and to the statistical receiver's security, the case in which $\mathsf{R}^*_{\mathsf{equiv}}$ plays the OT protocol with a pair $(q_b, q_{\bar{b}})$ and then is able to compute randomness $\mathsf{ran}_{\mathsf{OT}}$ and a different pair $((q'_b, q_{\bar{b}})$ that are still consistent with the transcript obtained in the OT execution, is statistically impossible . By the statistical receiver security of the OT protocol, H_3 and H_4 are statistically indistinguishable.

H_5. This is the ideal world experiment $\mathbf{Exp}^{\mathcal{S}}_{\mathsf{R}^*_{\mathsf{equiv}}}$.

□

□

2.3.2 UC-security Proof

In this section we show that protocol $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ depicted in Figure 5.3 is UC-secure, by showing a PPT ideal world adversary Sim such that for all PPT environment z, the view of the environment in the ideal process is indistinguishable from the view of the environment in the real process, in the $\mathcal{F}_{\mathsf{PUF}}$ hybrid model. Due to the straight-line extractability of $\mathsf{Com}_{\mathsf{shext}}$ and to the straight-line extractability and equivocality of $\mathsf{Com}_{\mathsf{equiv}}$, showing such a simulator Sim is almost straightforward.

Receiver is corrupt. Let $\mathsf{R}^*_{\mathsf{uc}}$ a malicious receiver. We show a PPT simulator Sim whose output is computational indistinguishable from the output obtained by $\mathsf{R}^*_{\mathsf{uc}}$ when interacting with the honest committer C_{uc}. The goal of Sim is to use the straight-line equivocator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ associated to protocol $\mathsf{Com}_{\mathsf{equiv}}$. To accomplish that, Sim has to force the output of the coin flipping, to the parameter generated by $\mathcal{S}_1$. Once this is done, then Sim can use $\mathcal{S}_2$ to complete the commitment phase, and $\mathcal{S}_3$ to equivocate the commitment. In order to force the output of the coin flipping, Sim extracts the commitment of α sent by $\mathsf{R}^*_{\mathsf{uc}}$ so that it can compute β appropriately. The extraction is done by running the extractor $E_{\mathsf{C}_{\mathsf{shext}}}$ associated to the protocol $\mathsf{Com}_{\mathsf{shext}}$.

Simulator 1.

Commitment Phase

- Run $(\bar{r}, \xi_1) \leftarrow \mathcal{S}_1(1^{ln})$.
- Execute protocol $\mathsf{Com}_{\mathsf{shext}}$ by running the associated extractor $E_{\mathsf{C}_{\mathsf{shext}}}$. If the output of the extractor is $\perp$, then abort. Else, let $\alpha^\star$ be the string extracted by $E_{\mathsf{C}_{\mathsf{shext}}}$. Set $\beta = \bar{r} \oplus \alpha^\star$, and send β to $\mathsf{R}^*_{\mathsf{uc}}$. If $\mathsf{R}^*_{\mathsf{uc}}$ aborts, then abort.
- When receiving the opening to α from $\mathsf{R}^*_{\mathsf{uc}}$, if the opening is not accepting, or if $\alpha \neq \alpha^\star$ then abort.
- Execute the commitment phase of protocol $\mathsf{Com}_{\mathsf{equiv}}$, on common input $\alpha \oplus \beta = \bar{r}$, by running $\mathcal{S}_2(\xi_1)$, and obtain ξ_2 as local output.

Decommitment Phase

- On input the bit b. Execute the decommitment phase of protocol $\mathsf{Com}_{\mathsf{equiv}}$ by running $\mathcal{S}_3(\xi_2, b)$.
- Output whatever R_{uc} outputs.

Lemma 5. *For all PPT real-world malicious receiver $\mathsf{R}^*_{\mathsf{uc}}$, for all PPT adversary z, it holds that:*

$$\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{com}}}_{\mathsf{Sim},z} \sim \mathsf{REAL}^{\mathcal{F}_{\mathsf{PUF}}}_{\mathsf{Com}_{\mathsf{uc}},\mathsf{R}^*_{\mathsf{uc}},\mathsf{z}}$$

Proof. It follows from the straight-line extractability of $\mathsf{Com}_{\mathsf{shext}}$ and from the straight-line equivocality of $\mathsf{Com}_{\mathsf{equiv}}$.

By the straight-line extractability of $\mathsf{Com}_{\mathsf{shext}}$ it holds that, with overwhelming probability, Sim obtains the value $\alpha^\star$ that will be later opened by $\mathsf{R}^*_{\mathsf{uc}}$, before it has to send the message β. Hence, Sim is able to force the output of the coin flipping to the value determined by $\mathcal{S}_1$. Then Sim just runs the simulator $\mathcal{S}_2$ in the commitment phase, and $\mathcal{S}_3$ in the decommitment phase. By the straight-line equivocality property of $\mathsf{Com}_{\mathsf{equiv}}$ the view generated by the interaction between $\mathsf{R}^*_{\mathsf{uc}}$ and Sim is computationally indistinguishable from the view generated by the interaction between $\mathsf{R}^*_{\mathsf{uc}}$ and an honest sender C_{uc}. □

Receiver and Committer are honest. In this case, z feeds the parties with their inputs, and activates the dummy adversary $\mathcal{A}$. $\mathcal{A}$ does not corrupt any party, but just observes the conversation between the committer and the receiver, forwarding every message to z.

In this case the simulator is almost equal to the simulator shown in Simulator 1 (when the receiver is corrupt). The only difference in this case is that, the receiver is also simulated by Sim. Therefore, Sim chooses both the strings used in the coin flipping by himself (α, β). Thus, there is no need for extraction.

More specifically, upon receiving the message $(\mathsf{receipt}, \mathsf{sid}, P_i, \mathsf{C}_{\mathsf{uc}})$ from $\mathcal{F}_{\mathsf{com}}$ in the ideal world, Sim draws a random tape to simulate the receiver, and runs the commitment phase as in Simulator 1, except for the second step. Instead of using the extractor associated to $\mathsf{Com}_{\mathsf{shext}}$ run by the receiver, Sim just picks values α and β so that $\bar{r} = \beta \oplus \alpha$ (where $\bar{r}$ is the value given in output by $\mathcal{S}_1$), and continues the commitment phase using such values. The decommitment phase is run identically to the decommitment phase of Simulator 1.

From the same argument of the previous case, the transcript provided by Sim is indistinguishable from the transcript provided by the dummy adversary $\mathcal{A}$ running with honest sender and receiver.

Committer is corrupt. In this case, the task of Sim is to extract the bit of the malicious committer $\mathsf{C}^*_{\mathsf{uc}}$ already in the commitment phase. This task is easily accomplished by running the straight-line extractor E_{equiv} associated to protocol $\mathsf{Com}_{\mathsf{equiv}}$. However, note that the binding property and thus the extractability property hold only when the common parameter $\bar{r}$ is uniformly chosen, while in protocol $\mathsf{Com}_{\mathsf{uc}}$ the common parameter is dictated by the coin flipping.

However, by the statistically hiding property of $\mathsf{Com}_{\mathsf{shext}}$, any unbounded adversary can not guess α better than guessing at random. Therefore for any $\mathsf{C}^*_{\mathsf{uc}}$ the distribution of $\alpha \oplus \beta$ is uniformly chosen over $\{0,1\}^{3nl}$, and thus the statistically binding property of $\mathsf{Com}_{\mathsf{equiv}}$ still holds.

Commitment Phase

- Pick a random α^{ln} and executes $\mathsf{Com}_{\mathsf{shext}}$ as the honest receiver.
- Obtain β from $\mathsf{C}^*_{\mathsf{uc}}$ and let $r = \alpha \oplus \beta$.
- Execute protocol $\mathsf{Com}_{\mathsf{equiv}}$ by running the associated extractor E_{equiv}. If the extractor aborts, abort. Else, let $b^\star$ the output of E_{equiv}. Send $(\mathsf{commit}, \mathsf{sid}, \mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}}, b^\star)$ to $\mathcal{F}_{\mathsf{com}}$

Lemma 6. *For all PPT real-world malicious committer $\mathsf{C}^*_{\mathsf{uc}}$, for all PPT adversary z, it holds that:*

$$\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{com}}}_{\mathsf{Sim},z} \sim \mathsf{REAL}^{\mathcal{F}_{\mathsf{PUF}}}_{\mathsf{Com}_{\mathsf{uc}},\mathsf{C}^*_{\mathsf{uc}},\mathsf{z}}$$

Proof. As mentioned before, the common input $\bar{r}$ computed through the coin-flipping, is uniformly distributed. Therefore the binding and the extractability property of $\mathsf{Com}_{\mathsf{equiv}}$ hold. The simulator runs protocol $\mathsf{Com}_{\mathsf{shext}}$ following the honest receiver, and runs the protocol $\mathsf{Com}_{\mathsf{equiv}}$ activating the straight-line extractor associated. By the simulation property of the extractor, the transcript generated by Sim is indistinguishable from the transcript generated by the honest receiver R_{uc}. From the extraction property satisfied by E_{equiv}, we have that Sim extracts the input bit of the adversary $\mathsf{C}^*_{\mathsf{uc}}$ and plays it in the ideal functionality, w.h.p.

□

- Chapter 3 -

Unconditional UC Commitments from (Physical) Setup Assumptions

Introduction

In this chapter we describe a tool for constructing UC-secure commitments given any straight-line extractable commitment. This essentially means that the task of constructing UC-secure commitment is reduced to the simpler task of achieving extractable commitments. This tool allows us to prove feasibility of unconditional UC-secure protocols (for a non-trivial functionality) in the stateless token model and in the malicious PUF model. More precisely, we provide a compiler that transforms any ideal extractable commitment – a primitive that we define – into a UC-secure commitment. An ideal extractable commitment is a statistically hiding, statistically binding and straight-line extractable commitment. The transformation uses the ideal extractable commitment as black-box and is unconditional, that is, it does not require any further assumption. The key advantage of such compiler is that, one can implement the ideal extractable commitment with the setup assumption that is more suitable with the application and the technology available.

We then provide an implementation of the ideal extractable commitment scheme in the malicious PUFs model introduced in Chapter 2 ([80]). By plugging our extractable commitment scheme in our compiler we obtain the first unconditional UC-secure commitment with malicious PUFs.

We then construct ideal extractable commitments using stateless tokens. We use some of the ideas employed for the PUF construction, but implement them with different techniques. Indeed, PUFs are intrinsically unpredictable, and even having oracle access to a PUF an unbounded adversary cannot predict the function run by it. With stateless tokens we do not have such guarantee and free access to a token might completely reveal the function embedded in it. Our protocol is secure in the standard stateless model, where the adversary has no restriction and can send malicious stateful tokens.

By plugging such protocol in our compiler, we achieve the first unconditional UC-secure commitment scheme with stateless tokens. Given that unconditional OT is im-

possible with stateless tokens [53], this result completes the picture concerning feasibility of unconditional UC-security with stateless tokens.

Remark 1. In the following (as it happens in all previous work on PUFs/tokens), it is assumed that even an unbounded adversary can query the PUF/token only a polynomial number of times. This assumption is necessary. Indeed, if we allowed the adversary to query the PUF/token on all possible challenges, then she can derive the truth table implemented by the physical device.

This chapter is organized as follows. Section 3.1 provides definition of extractable commitment and recall the stateless token model. In Section 3.2 we present our compiler, that we instantiate with PUFs and tokens using the ideal extractable commitments presented respectively in Section 3.3 and Section 3.4. We conclude this chapter with discussions about optimization and reusing of tokens/PUFs.

3.1 Definitions

This section presents the definition Ideal Extractable Commitment based on (physical) assumptions. Moreover, the UC-formulation of stateless tokens due to [62] and [23] is recalled in Section 3.1.2

3.1.1 Ideal Extractable Commitment Scheme

We denote by $\mathcal{F}_{\mathsf{aux}}$ the ideal set-up functionality used by a real world protocol.

Definition 16 (Ideal Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ model)**.** *A commitment scheme is a tuple of PPT algorithms* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *implementing the following two-phase functionality, having access to an ideal set-up functionality* $\mathcal{F}_{\mathsf{aux}}$*. Given to* C *an input* $b \in \{0,1\}$*, in the first phase called* commitment phase*,* C *interacts with* R *to commit to the bit* b*. We denote this interaction by* $((\mathbf{c}, d), \mathbf{c}) \leftarrow \langle \mathsf{C}(\mathtt{com}, b), \mathsf{R}(\mathtt{recv}) \rangle$ *where* $\mathbf{c}$ *is the transcript of the (possibly interactive) commitment phase and* d *is the decommitment data. In the second phase, called* decommitment phase*,* C *sends* (b, d) *and* R *finally outputs "*accept*"* *or "*reject*"* *according to* $(\mathbf{c}, d, b)$*.* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *is an* ideal *commitment scheme if it satisfies the following properties.*

Completeness. *For any* $b \in \{0,1\}$*, if* C *and* R *follow their prescribed strategy then* R *accepts the commitment* $\mathbf{c}$ *and the decommitment* (b, d) *with probability 1.*

Statistically Hiding. *For any malicious receiver* R^* *the ensembles* $\{\mathsf{view}_{\mathsf{R}^*}\ (\mathsf{C}(\mathtt{com}, 0),\ \mathsf{R}^*)\ (1^n)\}_{n\in\mathbb{N}}$ *and* $\{\mathsf{view}_{\mathsf{R}^*}(\mathsf{C}(\mathtt{com},\ 1),\ \mathsf{R}^*)\ (1^n)\}_{n\in\mathbb{N}}$ *are statistically indistinguishable, where* $\mathsf{view}_{\mathsf{R}^*}\ (\mathsf{C}(\mathtt{com}, b),\ \mathsf{R}^*)$ *denotes the view of* R^* *restricted to the commitment phase.*

Statistically Binding. *For any malicious committer* C^**, there exists a negligible function* ϵ*, such that* C^* *succeeds in the following game with probability at most* $\epsilon(n)$*:*

On security parameter 1^n, C^* interacts with R in the commitment phase obtaining the transcript $\mathbf{c}$. Then C^* outputs pairs $(0, d_0)$ and $(1, d_1)$, and succeeds if in the decommitment phase, $\mathsf{R}(\mathbf{c}, d_0, 0) = \mathsf{R}(\mathbf{c}, d_1, 1) = \mathsf{accept}$.

We call the commitment scheme **ideal** *since both binding and hiding must hold against unbounded adversaries.*

Definition 17 (Interface Access to an Ideal Functionality $\mathcal{F}_{\mathsf{aux}}$)**.** *Let $\Pi = (P_1, P_2)$ be a two-party protocol in the $\mathcal{F}_{\mathsf{aux}}$-hybrid model. That is, parties P_1 and P_2 need to query the ideal functionality $\mathcal{F}_{\mathsf{aux}}$ in order to carry out the protocol. An algorithm M has* interface access *to the ideal functionality $\mathcal{F}_{\mathsf{aux}}$ w.r.t. protocol Π if all queries made by either party P_1 or P_2 to $\mathcal{F}_{\mathsf{aux}}$ during the protocol execution are intercepted (but not answered) by M, and M has oracle access to $\mathcal{F}_{\mathsf{aux}}$. Such queries are then forwarded to $\mathcal{F}_{\mathsf{aux}}$ and the answers are sent to the party. Namely, $\mathcal{F}_{\mathsf{aux}}$ can be a non programmable and non PPT functionality.*

Definition 18 (Ideal Extractable Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ model)**.** $\mathsf{IdealExtCom} = (\mathsf{C}_{\mathsf{ext}}, \mathsf{R}_{\mathsf{ext}})$ *is an* ideal extractable commitment scheme in the $\mathcal{F}_{\mathsf{aux}}$ model *if $(\mathsf{C}_{\mathsf{ext}}, \mathsf{R}_{\mathsf{ext}})$ is an ideal commitment and there exists a straight-line strict polynomial-time extractor E having* interface access *to $\mathcal{F}_{\mathsf{aux}}$, that runs the commitment phase only and outputs a value $b^\star \in \{0, 1, \perp\}$ such that for all malicious committer C^* the following properties are satisfied.*

- **Simulation:** *the view generated by the interaction between E and C^* is identical to the view generated when C^* interacts with the honest receiver $\mathsf{R}_{\mathsf{ext}}$:* $\mathsf{view}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{C}^*}(\mathsf{C}^*(\mathtt{com}, \cdot), \mathsf{R}_{\mathsf{ext}}(\mathtt{recv})) \equiv \mathsf{view}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{C}^*}(\mathsf{C}^*(\mathtt{com}, \cdot), E)$
- **Extraction:** *let $\mathbf{c}$ be a valid transcript of the commitment phase run between C^* and E. If E outputs $\perp$ then probability that C^* will provide an accepting decommitment is negligible.*
- **Binding:** *if $b^\star \neq \perp$, then probability that C^* decommits to a bit $b \neq b^\star$ is negligible.*

3.1.2 Ideal Functionality for Stateless Tokens

The original work of Katz [62] introduces the ideal functionality $\mathcal{F}_{\mathsf{wrap}}$ to model stateful tokens in the UC-framework. A stateful token is modeled as a Turing machine. In the ideal world, a party that wants to create a token, sends the Turing machine to $\mathcal{F}_{\mathsf{wrap}}$. The adversary is, of course, allowed to send an arbitrarily malicious (still PPT) Turing machine to $\mathcal{F}_{\mathsf{wrap}}$. This translates in the fact that the adversary can send a malicious token to the honest party. $\mathcal{F}_{\mathsf{wrap}}$ will then run the machine (keeping the state), when the designed party will ask for it. The same functionality can be adapted to model stateless tokens. It is sufficient that the functionality does not keep the state between two executions.

One technicality of the model proposed by [62] is that it assumes that the adversary knows the code of the tokens that she sends. In real life, this translates to the fact

that an adversary cannot forward tokens received from other parties, or tamper its own token, so that the actual behavior of the token is not known to anyone. The advantage of this assumption, is that in the security proof the simulator can *rewind* the token.

In [23], Chandran, Goyal and Sahai, modify the original model of Katz, so to allow the adversary to create tokens without knowing the code. Formally, this consists in changing the 'create' command of the $\mathcal{F}_{\mathsf{wrap}}$ functionality, which now takes as input an Oracle machine instead of a Turing machine. The model of [23] is even stronger and allows the adversary to encapsulate tokens.

Our security proofs are unconditional, and our simulator and extractor only exploit the interface access to the ideal functionality $\mathcal{F}_{\mathsf{wrap}}$ (i.e., they only observe the queries made by the adversary), namely, they do not need adversary's knowledge of the code. Therefore, our proofs hold in both [23] and [62] models. In this work, similarly to all previous work on stateless tokens [67, 54, 25], and also [53], we do not consider adversaries that can perform token encapsulation.

A simplification of the $\mathcal{F}_{\mathsf{wrap}}$ functionality as shown in [23] (that is very similar to the $\mathcal{F}_{\mathsf{wrap}}$ of [62]) is depicted in Fig. 3.1.

Functionality $\mathcal{F}_{\mathsf{wrap}}$

$\mathcal{F}_{\mathsf{wrap}}$ is parameterized by a polynomial $p(\cdot)$ and an implicit security parameter n.

Create. Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathcal{T})$ from P_i, where P_j is another party in the system and $\mathcal{T}$ is an *oracle* machine do:

1. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathcal{T})$ to P_j.
2. Store $(P_i, P_j, \mathcal{T})$.

Execute. Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, \mathcal{T}, msg)$ from P_j, find the unique stored tuple $(P_i, P_j, \mathcal{T})$ (if no such tuple exists, then do nothing). Run $\mathcal{T}$ with input msg for at most $p(n)$ steps and let out be the response (set $out = \perp$ if $\mathcal{T}$ does not respond in the allotted time). Send (sid, P_i, out) to P_j.

Figure 3.1: *The $\mathcal{F}_{\mathsf{wrap}}$ functionality.*

3.2 UC-secure Commitments from Ideal Extractable Commitments

In this section we show how to transform any ideal extractable commitment scheme into a protocol that UC-realizes $\mathcal{F}_{\mathsf{com}}$ functionality, *unconditionally*. Such transformation is based on the following building blocks.

Extractable Blobs. "Blob" was used in [13] to denote a commitment. In this paper we use the term blob to denote a *pair* of bit commitments, that however still represent

the commitment of one bit. The bit committed in a blob is the xor of the bits committed in the pair. As we shall see soon, the representation of a bit as its exclusive-or, allows to prove equality of the bits committed in two blobs without revealing the values and more importantly using commitments as black boxes. Let IdealExtCom be any ideal extractable commitment scheme satisfying Definition 18. If the commitment phase of IdealExtCom is interactive then the blob is the pair of transcripts obtained from the interaction. Procedures to create a blob of a bit b, and to reveal the bit committed in the blob, are the following.

Blob(b): Committer picks bits b^0, b^1 uniformly at random such that $b = b^0 \oplus b^1$. It commits to b^0, b^1 (in parallel) running IdealExtCom as sub-routine and obtains commitment transcripts $\mathbf{c}^0, \mathbf{c}^1$, and decommitments d^0, d^1. Let $\mathbf{B} = (\mathbf{c}^0, \mathbf{c}^1)$ the **blob** of b.

OpenBlob($\mathbf{B}$): Committer sends $(b^0, d^0), (b^1, d^1)$ to Receiver. Receiver accepts iff d^0, d^1 are valid decommitments of b^0, b^1 w.r.t. transcripts $(\mathbf{c}^0, \mathbf{c}^1)$ and computes $b = b^0 \oplus b^1$.

Clearly, a blob inherits the properties of the commitment scheme used as sub-protocol. In particular, in the above case, since IdealExtCom is used as sub-routine, each blob is statistically hiding/binding, and straight-line extractable.

Equality of Blobs. Given the representation of a bit commitment as a blob, a protocol due to Kilian [64] allows to prove that two committed bits (two blobs) are equal, without revealing their values. We build upon this protocol to construct a "simulatable" version, meaning that (given some trapdoor) a simulator can prove equality of two blobs that are *not* equal. Let $\mathbf{B}_i, \mathbf{B}_j$ be two blobs. Let $b_i = (b_i^0 \oplus b_i^1)$ be the bit committed in $\mathbf{B}_i$ and $b_j = (b_j^0 \oplus b_j^1)$be the bit committed in $\mathbf{B}_j$. Let P denote the prover and V the verifier. P proves to V that $\mathbf{B}_i$ and $\mathbf{B}_j$ are the commitment of the same bit (i.e., $b_i = b_j$). In this protocol we crucially use the extractability property of the underlying ideal extractable commitment scheme.

ProveBlobsEquality($\mathbf{B}_i, \mathbf{B}_j$)

1. V uniformly chooses $e \in \{0, 1\}$ and commits to e using IdealExtCom.
2. P sends $b = b_i^0 \oplus b_j^0$ to V.
3. V reveals e to P.
4. P reveals b_i^e and b_j^e (i.e., P sends decommitments d_i^e, d_j^e to V). V accepts iff $b = b_i^e \oplus b_j^e$.

The completeness of the protocol, follows from the completeness of the commitment scheme IdealExtCom used to commit the challenge e and to compute blobs $\mathbf{B}_i, \mathbf{B}_j$.

Properties of protocol ProveBlobsEquality**.** Protocol ProveBlobsEquality satisfies the following properties. (In the following discussion we use b_i (or b_j) meaning the "bit committed in $\mathbf{B}_i$ (or $\mathbf{B}_j$)".

Soundness: if $b_i \neq b_j$, any malicious prover P^* convinces V with probability $1/2$, that is the probability of guessing the challenge e. Here we are using the statistically hiding property of the ideal commitment IdealExtCom used to commit e. The formal proof is provided in Lemma 7.

Privacy: If $b_i = b_j$ then after executing the protocol, the view of any verifier V^*, is independent of the actual value of b_i, b_j (given that $\mathbf{B}_i, \mathbf{B}_j$ were secure at the beginning of the protocol). This claim is proved in Lemma 7.

Simulation: there exists a straight-line strictly PPT simulator SimFalse such that, for any $(\mathbf{B}_i, \mathbf{B}_j)$ that are not equal, i.e., $b_i \neq b_j$, for any malicious verifier V^*, produces a view that is statistically close to the case in which $(\mathbf{B}_i, \mathbf{B}_j)$ are equal, i.e., $b_i = b_j$ and V^* interacts with the honest P. The formal proof is provided in Lemma 9. Note that the protocol uses blobs in a black-box way. Note also, that a blob can be involved in a single proof only.

Lemma 7 (Soundness of ProveBlobsEquality)**.** *If* IdealExtCom *is an ideal commitment, then for any malicious prover P^*, there exists a negligible function ϵ, such that if $b_i \neq b_j$,* $\mathsf{Pr}[\, V accepts \,] = 1/2 + \epsilon$.

Proof. The prover can cheat in two ways: 1) by guessing the challenge. In this case P^* can just compute b as $b_i^e \oplus b_j^e$ and convince the verifier; 2) by breaking the binding of IdealExtCom used to compute the blobs. Due to the statistically hiding property of IdealExtCom, probability that any P^* guesses the challenge committed by V, is only negligibly better than $1/2$. Due to the statistically binding property of IdealExtCom, probability that P^* opens a commitment adaptively on the challenge is negligible. □

Lemma 8 (Privacy of ProveBlobsEquality)**.** *Assume that $\mathbf{B}_i, \mathbf{B}_j$ are statistically hiding commitments. If $b_i = b_j$ then for any malicious verifier V^* the view is independent on the actual value of b_i and b_j.*

Proof. We prove that given a view of V^*, any value for b_i, b_j is equally likely. The view of V^* after the execution of protocol ProveBlobsEquality consists of: $\mathbf{B}_i, \mathbf{B}_j, b, b_i^e, b_j^e$. We argue that any bit $\beta \in \{0,1\}$ is consistent with such view. Indeed, since bits $b_i^0, b_i^1, b_j^0, b_j^1$ are randomly chosen, for any bit β there exists a pair $b_i^{\bar{e}}, b_j^{\bar{e}}$ such that $b = b_i^{\bar{e}} \oplus b_j^{\bar{e}}$ and $\beta = b_i^e \oplus b_i^{\bar{e}}$ and $\beta = b_j^e \oplus b_j^{\bar{e}}$. □

Lemma 9 (Simulation of ProveBlobsEquality in the $\mathcal{F}_{\mathsf{aux}}$ model)**.** *If* IdealExtCom *is a straight-line extractable commitment in the $\mathcal{F}_{\mathsf{aux}}$-hybrid model, then there exists a straight-line PPT algorithm* SimFalse*, called simulator, such that for any V^*, the view of V^* interacting with* SimFalse *on input a pair $(\mathbf{B}_i, \mathbf{B}_j)$ of possibly not equal blobs (i.e., $b_i \neq b_j$) is statistically close to the view of V^* when interacting with P and $b_i = b_j$.*

Proof. In the following we use the assumption that blobs are statistically hiding, therefore given $\mathbf{B}_i, \mathbf{B}_j$, any pair b_i, b_j is equally likely to be the committed values. Let E be the straight-line extractor associated to $\mathsf{IdealExtCom}$ as required by Definition 18. On common input $(\mathbf{B}_i, \mathbf{B}_j)$, $\mathsf{SimFalse}$ has interface access to $\mathcal{F}_{\mathsf{aux}}$ and works as follows.

$\mathsf{SimFalse}$ $(\mathbf{B}_i, \mathbf{B}_j)$

1. (V^* has to commit to the challenge e.) For the commitment phase of $\mathsf{IdealExtCom}$, run extractor E as-subroutine forwarding all the messages computed by E to V^* and viceversa, and having interface access to $\mathcal{F}_{\mathsf{aux}}$ (access to $\mathcal{F}_{\mathsf{aux}}$ is needed to run procedure E). After the completion of the commitment phase, obtain $b^\star \in \{0, 1, \perp\}$ form E. If V^* or E aborts, then halt.
2. Send $b = b_i^{b^\star} \oplus b_j^{b^\star}$ to V^*. If $b^\star = \perp$ send a random bit.
3. Upon receiving the decommitment e of the challenge:
   - If $e \neq b^\star$ then abort. We call this event *extraction abort.*
   - Else, if $b^\star = \perp$ halt. Otherwise, reveal $b_i^{b^\star}$, $b_j^{b^\star}$.

Since E is straight-line (due to the straight-line extractability of $\mathsf{IdealExtCom}$) and generates a transcript that is identical to the one generated by an honest receiver (due to the simulation property of $\mathsf{IdealExtCom}$), the only deviation of $\mathsf{SimFalse}$ w.r.t. to an honest prover is in the computation of bit b. In the honest execution b is always $b_i^0 \oplus b_j^0$, in the simulated execution b depends on the challenge extracted, i.e., $b = b_i^{b^\star} \oplus b_j^{b^\star}$. For simplicity, let us assume that the challenge extracted $b^\star$ corresponds to the one that is later revealed by V^*, i.e.,$b^\star = e$ (we handle the case in which is not later).

We argue that, for any V^* the view obtained interacting with an honest prover P and $b_i = b_j$ (honest execution), is statistically close to the view obtained interacting with $\mathsf{SimFalse}$ and $b_i \neq b_j$ (simulated execution).

The view of V^* at the end of the execution of $\mathsf{ProveBlobsEquality}$ consists of: $((\mathbf{B}_i, \mathbf{B}_j), b, b_i^e, b_j^e)$. In case $e = 0$, it is easy to see that, given that blobs are statistically hiding, the view of the honest execution is statistically close to the view of the simulated execution. Indeed, in this case b is computed as $b_i^0 \oplus b_j^0$, exactly as in the honest execution.

In case $e = 1$, in the simulated experiment b is computed as $b_i^1 \oplus b_j^1$, deviating from the honest procedure where $b = b_i^0 \oplus b_j^0$. Here is sufficient to observe that, in the honest execution, $b_i = b_j$ therefore it holds that $b = b_i^1 \oplus b_j^1 = b_i^0 \oplus b_j^0$. Thus, distribution of (b, b_i^1, b_j^1) obtained in the simulation is still statistically close (given the hiding of blobs) to the distribution obtained from the honest execution.

When the challenge extracted (if any) is different from the one revealed by V^*, $\mathsf{SimFalse}$ aborts. Thus probability of observing abort in the simulated execution is higher than in the honest execution. Nevertheless, due to the extractability property of $\mathsf{IdealExtCom}$, probability of aborting because of extraction failure is negligible. □

Here we prove another property of ProveBlobsEquality that will be useful when proving the straight-line equivocality of protocol UCComCompiler. The following lemma is required only for the case in which the simulator was used to prove a false theorem (i.e., $b_i \neq b_j$). Indeed, when $b_i = b_j$ the transcript of the simulation is always statistically close to the transcript of the honest execution even after one of the blob is revealed.

Lemma 10 (Indistinguishability of the Simulation after one blob is revealed.)**.** *The view of V^* in the simulated execution (where $b_i \neq b_j$) is statistically close to the view of V^* in the honest execution (where $b_i = b_j$) even if, at the end of the protocol, one blob is revealed.*

Proof. Assume wlog that after the execution of ProveBlobsEquality, the value b_i of blob $\mathbf{B}_i$ is reveal. This means that both bits b_i^0, b_i^1 are revealed. The view of V^* at this point consists of values (b, b_j^e, b_i^0, b_i^1). So only bit $b_j^{\bar{e}}$ is not revealed. Now consider again the honest experiment, when $b_i = b_j$ and $b = b_i^0 \oplus b_j^0$, and the simulated experiment where $b_i \neq b_j$ and $b = b_i^e \oplus b_j^e$. We want to argue that, even after b_i is known, still the view generated by the simulator is statistically close to the view of the honest execution. Consider the case in which $e = 1$ (the case in which $e = 0$ follows straight-forwardly). At the beginning all four bits $b_i^0, b_i^1, b_j^0, b_j^1$ are hidden to V^*. After the protocol execution V^* knows bit b_i^1, b_j^1 and b that is *supposed to be* xor of b_i^0, b_j^0. We already proved that in this case any value b_i, b_j of the blobs is equally likely. After blob $\mathbf{B}_i$ and therefore bit b_i is revealed, V^* knows 3 out of 4 bits, and the value of b_j^0 is determined by the knowledge of b_i. Indeed, if $b_i = b_j$ then $b_j^0 = b_i \oplus b_j^1$. Furthermore, since $b = b_i^0 \oplus b_j^0$, the values of b_j^0 must satisfy also condition $b_j^0 = b \oplus b_i^0$. Hence, $b_i \oplus b_j^1 = b \oplus b_i^0$. In the honest executions the equation is certainly satisfied since $b_i = b_j$ and b is honestly computed. We show that in the simulated experiment, the equation always holds (note that in this argument we are using the fact that all shares $b_i^0, b_i^1, b_j^0, b_j^1$ are randomly chosen). Given the equation:

$$b_i \oplus b_j^1 = b \oplus b_i^0$$

given that in the simulation $b = b_i^1 \oplus b_j^1$, and $b_i = b_i^0 \oplus b_i^1$; by replacing b and b_i we have:

$$b_i^0 \oplus b_i^1 \oplus b_j^1 = b_i^1 \oplus b_j^1 \oplus b_i^0$$

□

3.2.1 The construction

We construct unconditional Universally Composable secure commitments using *extractable* blobs and protocol ProveBlobsEquality as building blocks.

We want to implement the following idea. The committer sends two blobs of the same bit and proves that they are equal running protocol ProveBlobsEquality. In the decommitment phase, it opens only one blob (a similar technique is used in [59], where instead the commitment scheme is crucially used in a non black-box way). The simulator would extract the bit of the committer by exploiting the extractability property of blobs.

The simulator can instead equivocate, by committing to the pair 0 and 1, and cheating in the protocol ProveBlobsEquality, by running the simulator associated to it. In the opening phase, it then opens the blob corresponding to the right bit.

This idea does not work straight-forwardly since soundness of protocol ProveBlobsEquality holds only with probability 1/2 and thus a malicious committer can break binding with the same probability. We cannot amplify the soundness by running many proofs on the same pair of blobs, since a blob can be involved in a proof only once. (This is due to the fact that we threat blobs in a black-box manner). Running many proofs among many independent pairs of blobs, and ask the committer to open half of them, is the way to go.

Specifically, the committer will compute n pairs of (extractable) blobs. Then it proves equality of each consecutive pair of blobs by running protocol ProveBlobsEquality with the receiver. The commitment phase is successful if all equality proofs are accepting. In the decommitment phase, the committer opens one blob for each pair. Namely, it reveals n blobs. Notice that, the committer cannot open any arbitrary set of blobs. The freedom of the committer is only in the choice of the index to open for each pair. The receiver accepts if two conditions are satisfied: 1) the committer opens one blob for each consecutive pair, 2) all revealed blobs open to the same bit. The construction is formally described in Fig. 3.2.

Protocol UCComCompiler
Committer's Input: $b \in \{0,1\}$.
Commitmen Phase

1. Committer: run $\mathsf{Blob}(b)$ $2n$ times. Let $\mathbf{B}_1, \ldots, \mathbf{B}_{2n}$ be the **blobs** obtained.
2. Committer $\Leftrightarrow$ Receiver: for $i = 1$; $i = i + 2$; $i \leq 2n - 1$; run $\mathsf{ProveBlobsEquality}(\mathbf{B}_i, \mathbf{B}_{i+1})$.
3. Receiver: if all equality proofs are accepting, accept the commitment phase.

Decommitment Phase

1. Committer: for $i = 1$; $i = i + 2$; $i \leq 2n - 1$; randomly choose $\ell \in \{i, i+1\}$ and run $\mathsf{OpenBlob}(\mathbf{B}_\ell)$ with the Receiver.
2. Receiver: 1) check if Committer opened one blob for each consecutive pair; 2) if all n blobs open to the same bit b, output b and accept. Else output reject.

Figure 3.2: UCComCompiler*: Unconditional UC Commitments from any Ideal Extractable Commitments.*

Theorem 4. *If* IdealExtCom *is an ideal extractable commitment scheme in the* $\mathcal{F}_{\mathsf{aux}}$*-hybrid model, then protocol in Fig. 3.2 is an unconditionally UC-secure bit commitment scheme in the* $\mathcal{F}_{\mathsf{aux}}$*-hybrid model.*

Proof Intuition. To prove UC-security we have to show a straight-line simulator Sim which correctly simulates the view of the real-world adversary $\mathcal{A}$ and extracts her input. Namely, when simulating the malicious committer in the ideal world, Sim internally runs the real-world adversarial committer $\mathcal{A}$ simulating the honest receiver to her, so to extract the bit committed to by $\mathcal{A}$, and play it in the ideal world. This property is called extractability. When simulating the malicious receiver in the ideal world, Sim internally runs the real-world adversarial receiver $\mathcal{A}$ simulating the honest committer to her, without knowing the secret bit to commit to, but in such a way that it can be opened as any bit. This property is called equivocality. In the following, we briefly explain why both properties are achieved. In the proof we assume that parties communicate through authenticated channels.

Straight-line Extractability. It follows from the straight-line extractability and binding of IdealExtCom and from the soundness of protocol ProveBlobsEquality. Roughly, Sim works as follows. It plays the commitment phase as an honest receiver (and running the straight-line extractor of IdealExtCom having access to $\mathcal{F}_{\mathsf{aux}}$). If all proofs of ProveBlobsEquality are *successful*, Sim extracts the bits of each consecutive pair of blobs and analyses them as follows. Let $b \in \{0,1\}$. If all extracted pairs of bits are either (b,b) or $(\bar{b},b)$, (i.e. there are no pairs like $(\bar{b},\bar{b})$), it follows that, the only bit that $\mathcal{A}$ can successfully decommit to, is b. In this case, Sim plays the bit b in the ideal world.

If there is at least a pair (b,b) and a pair $(\bar{b},\bar{b})$, then $\mathcal{A}$ cannot provide any accepting decommitment (indeed, due to the binding of blobs, $\mathcal{A}$ can only open the bit b from one pair, and the bit $\bar{b}$ from another pair, thus leading the receiver to reject). In this case Sim sends a random bit to the ideal functionality.

If all the pairs of blobs are not equal, i.e., all pairs are either $(\bar{b},b)$ or $(b,\bar{b})$, then $\mathcal{A}$ can later decommit to any bit. In this case the simulator fails in the extraction of the bit committed, and aborts. Note that, this case happens only when all the pairs are not equal. Thus $\mathcal{A}$ was able to cheat in all executions of ProveBlobsEquality. Due to the soundness of ProveBlobsEquality, this event happens with probability 2^{-n}.

Straight-line Equivocality. It follows straight-forwardly from the simulation property of ProveBlobsEquality. Sim in this case works as follows. It prepares n pairs of blobs such that each pair contains blob of 0 and blob of 1, in randomly chosen positions. Sim is able to cheat in all executions of ProveBlobsEquality, by running the straight-line simulator associated to this protocol. In the decommitment phase, after having received the bit to decommit to, for each pair, Sim reveals the blob corresponding to the right bit.

Note that, in both cases, Sim crucially uses the extractor associated to IdealExtCom, that in turn uses the access to $\mathcal{F}_{\mathsf{aux}}$. The formal proof of Theorem 4 is provided in Section 3.2.2.

In Section 3.3 we show an implementation of IdealExtCom with malicious PUFs, while in Section 3.4, we show how to implement IdealExtCom using stateless token. By plugging such implementations in protocol UCComCompiler we obtain the first unconditional UC-secure commitment scheme with malicious PUFs (namely, in the $\mathcal{F}_{\mathsf{PUF}}$-hybrid model), and stateless tokens (namely, in the $\mathcal{F}_{\mathsf{wrap}}$-hybrid model).

3.2.2 UC-security Proof

In this section we provide formal proof of Theorem 4. We show a straight-line simulator Sim having interface access to $\mathcal{F}_{\mathsf{aux}}$ and interacting with $\mathcal{F}_{\mathsf{com}}$ only, that for any environment z, generates a transcript that is indistinguishable from the transcript that z obtains from the real-world adversary $\mathcal{A}$ participating (or just observing) the real protocol execution. We distinguish three cases, according to which party z corrupts, if any.

Committer and Receiver are honest. In this case the real-world adversary $\mathcal{A}$ is instructed by z to not corrupt any party. The goal of the simulator is to generate the transcript of the interaction between honest parties $\mathsf{C_{uc}}, \mathsf{R_{uc}}$. The procedure of Sim is described in Simulator 2.

Simulator 2. *[*Sim *in the honest-honest case.]*

Commitment Phase.
Whenever $\mathcal{F}_{\mathsf{com}}$ writes $(\mathsf{receipt}, \mathsf{sid}, \mathsf{C_{uc}}, \mathsf{R_{uc}})$ to the communication tape of Sim in the ideal world, then this message indicates that z wrote the secret bit b to the input tape of $\mathsf{C_{uc}}$. Sim simulates the transcript of the commitment phase between $\mathsf{C_{uc}}$ and $\mathsf{R_{uc}}$ as follows.

1. *For* $(i = 1;\ i = i + 2;\ i \leq 2n - 1)$*:*
   - *pick randomly* $\ell_i^0 \in \{i, i+1\}$*; let* $\ell_i^1 \leftarrow \{i, i+1\}/\{\ell_i^0\}$.
   - *let* $\mathsf{C_{uc}}$ *run* $\mathbf{B}_{\ell_i^0} = \mathsf{Blob}(0)$ *and* $\mathbf{B}_{\ell_i^1} = \mathsf{Blob}(1)$ *with* $\mathsf{R_{uc}}$.

   When the simulated $\mathsf{C_{uc}}$ *or* $\mathsf{R_{uc}}$ *queries functionality* $\mathcal{F}_{\mathsf{aux}}$*, interact with* $\mathcal{F}_{\mathsf{aux}}$ *from their behalf.*

2. *For* $(i = 1;\ i = i+2;\ i \leq 2n-1)$*, simulate execution of* $\mathsf{ProveBlobsEquality}(\bar{\mathbf{B}}_i, \bar{\mathbf{B}}_{i+1})$ *as follows (the following steps correspond to procedure* $\mathsf{SimFalse}$ *except for the first step, in which the challenge is not extracted but randomly chosen by* Sim*):*
   - *pick a random challenge* e*, and let* $\mathsf{C_{uc}}, \mathsf{R_{uc}}$ *run commitment phase of* $\mathsf{IdealExtCom}$ *where* $\mathsf{R_{uc}}$ *runs as a committer on input* e*, and* $\mathsf{C_{uc}}$ *runs as a receiver.*
   - *write* $b = b_i^e \oplus b_{i+1}^e$ *on* $\mathsf{R_{uc}}$*'s communication tape.*
   - *write the decommitment of* e *on* $\mathsf{C_{uc}}$*'s communication tape.*
   - *write decommitments of* b_i^e, b_{i+1}^e *on the communication tape of* $\mathsf{R_{uc}}$.

   In any of the steps above, delay or to drop a message according to the strategy of the real-world adversary $\mathcal{A}$.

Decommitment phase.
When receiving $(\mathsf{open}, \mathsf{sid}, \mathsf{C_{uc}}, \mathsf{R_{uc}}, b)$ *simulate the transcript of the decommitment phase as follows.*

1. *If* $b = 0$ *then for* $(i = 1;\ i = i + 2;\ i \leq 2n - 1)$ *run* $\mathsf{OpenBlob}(\mathbf{B}_{\ell_i^0})$.

2. *If* $b = 1$ *then for* $(i = 1; i = i + 2; i \leq 2n - 1)$ *run* $\mathsf{OpenBlob}(\mathbf{B}_{\ell_i^1})$.

Note that, in Step 2, Sim is basically running algorithm $\mathsf{SimFalse}$. The only difference with $\mathsf{SimFalse}$ is that the challenge e is not extracted using extractability of $\mathsf{IdealExtCom}$, but it is chosen by Sim itself. Therefore, in the following proof we will use the lemmata proved in Section 3.2.

Claim 3.2.1 (Indistinguishability of the simulation when both parties are honest)**.** *If blobs are ideal commitments, for any real-world adversary* $\mathcal{A}$ *and any environment* z*, the transcript generated by* Sim *(Simulator 2) is statistically indistinguishable from the interaction between honest real-world* $\mathsf{C_{uc}}, \mathsf{R_{uc}}$.

Proof. In this proof we use only the statistically hiding property of $\mathsf{IdealExtCom}$ commitment scheme used to implement the Blob procedure, and the interface access of Sim to $\mathcal{F}_{\mathsf{aux}}$ which is necessary to honestly execute protocol $\mathsf{IdealExtCom}$.

In the honest-honest case, the environment z sets the input of the honest sender $\mathsf{C_{uc}}$, observes the communication between $\mathsf{C_{uc}}$ and $\mathsf{R_{uc}}$, and possibly delays/drops messages (we assume authenticated channel) of the protocol through the dummy adversary $\mathcal{A}$. We show that the transcript simulated by Sim 2 is statistically close to the actual transcript obtained from the real interaction of honest $\mathsf{C_{uc}}, \mathsf{R_{uc}}$. The proof goes by hybrids arguments. It starts from the real world, hybrid H_0, in which $(\mathsf{C_{uc}}, \mathsf{R_{uc}})$ honestly run the protocol using the input received from z, and it ends to the ideal world, hybrid H_4, where Sim simulates both parties without knowing the actual input.

Hybrid H_0**:** This is the real world.

Hybrid H_1**:** In this hybrid, consider simulator Sim_1. Sim_1 obtains the input b chosen by z for $\mathsf{C_{uc}}$, it honestly runs procedure of $\mathsf{C_{uc}}$ on input b and procedure $\mathsf{R_{uc}}$, using independently random tapes (and forwarding the queries of $\mathsf{C_{uc}}, \mathsf{R_{uc}}$ to the ideal functionality $\mathcal{F}_{\mathsf{aux}}$ when they run the extractable commitment scheme). In addition, Sim_1 internally simulates a copy of the dummy adversary $\mathcal{A}$ as well as $\mathcal{A}$'s communication with z, and let $\mathcal{A}$ control the scheduling of the communication. H_1 is just the real world protocol, executed through the simulator Sim_1. Clearly, hybrids H_0 and H_1 are identical.

Hybrid H_2^j (for $1 \leq j \leq n$): The difference between hybrid H_2^j and hybrid H_2^{j-1} is that in Hybrid H_2^j, the j-th instance of Protocol $\mathsf{ProveBlobsEquality}$, is simulated. Specifically, in hybrid H_2^j, Sim_2^j simulates the j-th instance of $\mathsf{ProveBlobsEquality}$ by running Step 4 of Sim 2 instead of running the honest prover procedure (as the honest $\mathsf{C_{uc}}$ would do).

We claim that the views obtained from hybrids H_2^{j-1} and H_2^j are statistically close.

In hybrid H_2^{j-1} the j-th execution of $\mathsf{ProveBlobsEquality}$ is executed following the procedure of the honest prover P. In hybrid H_2^j, the procedure of a modified (the challenge e do not need to be extracted) $\mathsf{SimFalse}$ is followed instead. By

lemma 8, it holds that the transcript generated by SimFalse is statistically close to the transcript generated by an honest prover. In our case is even identical since we do not have to consider the negligible probability of failure of the extraction, and since the pair of blob $\mathbf{B}_j, \mathbf{B}_{j-1}$ are equal.

Hence, hybrids H_2^{j-1} and H_2^j are identical.

Note that, Hybrid H_2^0 corresponds to the real experiment H_1 where all proofs are given by honestly running the prover of ProveBlobsEquality, and H_2^n corresponds to the case in which all proof are simulated, by running SimFalse.

Hybrid H_3: In this hybrid, we consider simulator Sim_3. In the commitment phase, Sim_3 chooses, for each i, the indexes ℓ_i^0, ℓ_i^1. Then in the decommitment phase Sim_3, pick a random bit d, and for each pair i, it opens always the blob in position ℓ_i^d. This hybrid is identical to H_2^n.

Hybrid H_4: In this hybrid, we consider simulator Sim_4. In the commitment phase Sim_4 follows Step 2 of Simulator 2. Namely, for all indexes ℓ_i^0 it commits (it "blobs") to 0, and it commits to 1 for the remaining index ℓ_i^1. Then in the decommitment phase, for each i it opens blobs in position ℓ_i^b. Note that here Sim_4 is not using the knowledge of the input b in the commitment phase.

The difference between hybrids H_3 and H_4 is that blobs do not commit to the same bit, they are not all equal. Therefore, in H_4 the simulated proofs are given on pairs of blobs that are not equal, and then one of the blobs is revealed. By Lemma 10, and the statistically hiding property of blobs (that are ideal commitment schemes) it follows that hybrids H_3 and H_4 are statistically close.

Noticing that Sim_4 corresponds to the procedure of Sim (Simulator 2), we have that hybrid H_4 is the ideal world. The claim is proved.

□

Receiver is corrupt. In this case the environment z instructs the real-world adversary $\mathcal{A}$ to corrupt the receiver R_{uc}. The simulator in this case, is very close to Simulator 2 shown for the honest-honest case. Therefore we will just point out the differences with the previous simulator, and how the same indistinguishability proof can be consequently adapted.

Concerning the simulator, the main difference with Simulator 2 is in Step 4. While in the honest-honest case the challenge is chosen by Sim 2 itself, in the malicious receiver case, the challenge must be extracted from the adversary. This simply means that Step 4 must be replaced with procedure SimFalse shown in Lemma 8. Furthermore, the simulator in this case is not simulating R_{uc}, but is internally running $\mathcal{A}$ that plays as a receiver. Thus, it has to take care of $\mathcal{A}$ aborting the protocol at any point.

The proof that such simulation is indistinguishable from the real-world execution goes along the same lines of the proof provided for the honest-honest case. The main difference is in hybrid H_2, that in case of malicious receiver, is only statistically close to hybrid

H_1. Indeed, when the receiver is malicious we have to consider the negligible probability of the failure of the extractor associated to the commitment scheme $\mathsf{IdealExtCom}$.

Committer is corrupt. In this case, the environment z instructs the adversary $\mathcal{A}$ to corrupt the sender $\mathsf{C_{uc}}$. The simulator Sim internally simulates a copy of the dummy adversary $\mathcal{A}$ as well as $\mathcal{A}$'s communication with z. In addition, Sim simulates the honest receiver $\mathsf{R_{uc}}$ to $\mathcal{A}$. The goal of Sim is to extract the bit that $\mathcal{A}$ is committing to in the simulated execution, so that it can send it to the ideal functionality $\mathcal{F}_{\mathsf{com}}$.

The procedure of Sim very roughly is the following. Sim extracts the bits committed in each blob by running the extractor of $\mathsf{IdealExtCom}$ and then executes protocols $\mathsf{ProveBlobsEquality}$ exactly as the honest receiver $\mathsf{R_{uc}}$. If all the executions of $\mathsf{ProveBlobsEquality}$ are *accepting*, then Sim looks at the extracted pair of bits, and proceeds as follow. If there exists at least one pair (b, b) and at least one pair $(\bar{b}, \bar{b})$, (for a bit b), then the adversary, that has to open at least one bit per pair, will open to b and $\bar{b}$, thus leading the receiver to reject. Indeed, the receiver expects that all bits opened are equal. Thus, in this case the adversary cannot successfully open to any bit. Hence, the simulator will play the bit 0 in the ideal functionality. If there exist only pairs in the form (b, b) or $(b, \bar{b})$, then the adversary, can successfully open only to bit b. In this case, Sim will play b in the ideal world. Finally, if all pairs are *not equal*, that is, each pair is either $(b, \bar{b})$ or $(\bar{b}, b)$, then the adversary can later successfully open to both b and $\bar{b}$. In this case, Sim has no clue on which bit to play in the ideal functionality and fails. Since this case happens when the adversary was able to prove equality of n pairs that are not equal, probability that the adversary passes all these false proofs is 2^{-n}, which is negligible. Thus, probability that Sim fails in the extraction of the secret bit, is negligible as well. Sim is formally defined in Simulator 3.

Simulator 3 (Sim in case sender $\mathsf{C_{uc}}$ is corrupt.)**.** Activate $\mathcal{A}$ on input the security parameter n and the secret bit received by z. When $\mathcal{A}$ starts the commitment phase, proceeds as follows.

Commitment Phase.

1. *For $j = 1, \ldots, 2n$: extract the bit committed in blob $\mathbf{B}_j$. Namely, run the procedure of the extractor E associated to* $\mathsf{IdealExtCom}$ *for the pair of commitments in $\mathbf{B}_j$. Obtain bits b_j^0,b_j^1 from the extraction. Set $b_j = b_j^0 \oplus b_j^1$.* In this phase Sim uses the interface access to $\mathcal{F}_{\mathsf{aux}}$ as required by E. If E aborts in any of the executions, then Sim also aborts. *If $\mathcal{A}$ does not abort in any of the commitments, proceeds to the next step.*

2. *If $\mathcal{A}$ proceeds to run* $\mathsf{ProveBlobsEquality}(\mathbf{B}_i, \mathbf{B}_{i+1})$*, for all adjacent pairs, then follow the procedure of the honest receiver.*

3. *If all proofs are successful, consider the bits extracted in Step 1, and check which case applies:*

(a) *There exists a bit b such all adjacent pairs of extracted bit are either (b, b) or $(b, \bar{b})$. In this case, since in the decommitment phase $\mathcal{A}$ is required to open one bit for each pair, there is only one bit that $\mathcal{A}$ can possibly decommit to, and is the bit b. Thus, send $(\mathsf{commit}, \mathsf{sid}, \mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}}, b)$ to $\mathcal{F}_{\mathsf{com}}$.*

(b) *There exists at least an adjacent pair of bits (b, b) and at least one pair of bits $(\bar{b}, \bar{b})$. In this case, $\mathcal{A}$ that has to open at least one bit for each pair, cannot successfully commit to any bit. Thus send $(\mathsf{commit}, \mathsf{sid}, \mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}}, 0)$ to $\mathcal{F}_{\mathsf{com}}$.*

(c) *(**Failure**) Each adjacent pair is either $(0, 1)$ or $(1, 0)$. In this case, $\mathcal{A}$ could correctly decommit to both 0 and 1. Thus, abort. We call this event* Input Extraction Failure.

Decommitment phase.
If $\mathcal{A}$ correctly decommits to a bit b, (i.e., all blobs revealed agree on the same value b), send $(\mathsf{open}, \mathsf{sid}, \mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}}, b)$ to $\mathcal{F}_{\mathsf{com}}$. Else, if $\mathcal{A}$ aborts, halt. If b is different from the one sent in the commitment phase, then abort. We call this even Binding Failure.

Claim 3.2.2 (Indistinguishability of the simulation when the sender is corrupt)**.** *If blobs are ideal* extractable *commitments, for any real-world adversary $\mathcal{A}$ corrupting the sender C_{uc}, any environment z, it holds that view $\mathsf{REAL}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{UCCom},\mathcal{A},\mathsf{z}}$ is statistically close to $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}\ 3,z}$.*

Proof. Sim 3 behaves almost identically to honest receiver R_{uc}. Indeed, it runs E in the first step, that due to the simulation property of IdealExtCom, generates a view that is identical to the one generated by an honest receiver. Then it honestly follows protocol ProveBlobsEquality. However, differently from the honest receiver, Sim 3 aborts more often. Specifically, Sim 3 additionally aborts in the following two cases:

> Case 1. In Step 1, when the extractor E fails in extracting the bit from any of the blobs.
>
> Case 2. In Step 3, Sim fails in determining the bit committed to by $\mathcal{A}$. We call this event *Input extraction Failure*, since Sim fails in extracting the input to send to the ideal functionality $\mathcal{F}_{\mathsf{com}}$.
>
> Case 3. In the decommitment phase $\mathcal{A}$ opens to a bit b that is different from the one extracted by Sim.

Due to the extractability property of the ideal extractable commitment IdealExtCom, Case 1 happens only with negligible probability. Due to Lemma 11, probability of Case 2 is also negligible. Finally, due to the statistically binding property of Blobs, probability that $\mathcal{A}$ can open to a bit that is different from the one extracted is negligible. Therefore, the view of $\mathcal{A}$ simulated by Sim is statistically close to the view obtained from the interaction with real world receiver. Which implies that the distribution of the input extracted by Sim is statistically close to the distribution of the input played in the real world, and the communication between $\mathcal{A}$ and z simulated by Sim is also statistically close to the communication of z with $\mathcal{A}$ interacting in the real protocol. Which implies that $\mathsf{REAL}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{UCCom},\mathcal{A},\mathsf{z}}$ and $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}\ 3,z}$ are statistically close. □

Lemma 11. *Probability of event* Input extraction Failure *is negligible.*

Proof. Event *Input extraction Failure* happens when *both* the following events happen:

> **Event 1:** all executions of protocol ProveBlobsEquality are successful. Namely, for all i [1], ProveBlobsEquality($\mathbf{B}_i$, $\mathbf{B}_{i+1}$) provided by $\mathcal{A}$ is accepting.
>
> **Event 2:** Each consecutive pair of blobs is not equal. Namely, for all i, $b_i \neq b_j$, where b_i and b_j are the bits committed respectively in $\mathbf{B}_i$, $\mathbf{B}_{i+1}$.

Due to the soundness of protocol ProveBlobsEquality, an adversary committing to n consecutive pairs that are all not equal, passes all the equality proof with probability $\frac{1}{2^n}$, which is negligible.

□

3.3 Ideal Extractable Commitments from (Malicious) PUFs

In this section we show a construction of ideal extractable commitment in the $\mathcal{F}_{\mathsf{PUF}}$ model. We first construct and ideal commitment scheme in the $\mathcal{F}_{\mathsf{PUF}}$ model that we denote by IdealComPuf. Then we transform this protocol into an extractable commitment scheme. For simplicity, in the informal description of the protocol, we omit mentioning the use of fuzzy extractors and the formalism for invoking the $\mathcal{F}_{\mathsf{PUF}}$ functionality. Such details are provided in the formal descriptions of Fig. 3.3 and Fig. 3.4.

Ideal Commitment Scheme in the $\mathcal{F}_{\mathsf{PUF}}$ Model. The idea behind the protocol IdealComPuf $= (\mathsf{C}_{\mathsf{pufIdeal}}, \mathsf{R}_{\mathsf{pufIdeal}})$, is to turn Naor's commitment scheme [76] [2] which is statistically binding but only computationally hiding, into statistically hiding and binding, by replacing the PRG with a (possibly *malicious*) PUF. Roughly, protocol IdealComPuf goes as follows.

At the beginning of the protocol, the committer creates a PUF, that we denote by $\mathcal{P}_S$. It preliminary queries $\mathcal{P}_S$ with a random string q to obtain the response σ_S, and finally delivers the PUF $\mathcal{P}_S$ to the receiver. After receiving the PUF, the receiver sends a random string r (i.e., the first round of Naor's commitment) to the committer. To commit to a bit b, the committer sends $\mathbf{c} = \sigma_S \oplus (r \wedge b)$ to the receiver. In the decommitment phase, the committer sends (b, s) to the receiver, who checks the commitment by querying $\mathcal{P}_S$ with s. Hiding intuitively follows from the fact that, a fuzzy extractor applied to the PUF-response σ_S, yields to a uniformly distributed value (this property is called Extraction Independence). Thus, commitment of 1, $\mathbf{c} = \sigma_S \oplus r$ and commitment of 0, $\mathbf{c} = \sigma_S$, are statistically close. Binding follows the same argument of Naor's scheme. The formal description of IdealComPuf is provided in Fig. 3.3. Protocol IdealComPuf

[1] for $(i = 1;\ i = i + 2;\ i < n)$

[2] Naor's scheme is a two-round commitment scheme. In the first round the receiver sends a random string $r \xleftarrow{\$} \{0,1\}^{3n}$ to the committer. In the second round, the committer picks a random string $s \xleftarrow{\$} \{0,1\}^n$, computes $y \leftarrow G(s)$ and sends $y \oplus (r \wedge b)$ to the receiver, where $G : \{0,1\}^n \to \{0,1\}^{3n}$ is a PRG and b is the bit to commit to.

is a bit commitment scheme, but it can be straight-forwardly transformed into string commitment by committing each bit of the string in parallel, reusing the same PUF.

Protocol $\mathsf{IdealComPuf}$
$\mathcal{F}_{\mathsf{PUF}}$ is parameterized with a PUF-family $\mathcal{P}=(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, \tilde{m})$. $(\mathsf{FuzGen}, \mathsf{FuzRep})$ is a $(\tilde{m}, \ell, t, \epsilon)$-fuzzy extractor of appropriate matching parameters such that $\ell = 3n$. $\mathcal{P}_S$ denote the id for accessing to the PUF created by the committer. Committer's Input: $b \in \{0,1\}$.

Commitment Phase

1. Committer $\mathsf{C}_{\mathsf{puflideal}}$: send $(\mathsf{init}_{\mathsf{PUF}}, \mathtt{normal}, \mathcal{P}_S, \mathsf{C}_{\mathsf{puflideal}})$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain response $(\mathsf{initialized}_{\mathsf{PUF}}, \mathcal{P}_S)$. Select a random string $q \in \{0,1\}^n$, send $(\mathsf{eval}_{\mathsf{PUF}}, \mathcal{P}_S, \mathsf{C}_{\mathsf{puflideal}}, q)$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain response $(\mathsf{response}_{\mathsf{PUF}}, \mathcal{P}_S, q, a)$. Compute $(st_S, p_S) \leftarrow \mathsf{FuzGen}(a)$, and send p_S to $\mathsf{R}_{\mathsf{puflideal}}$ and $(\mathsf{handover}_{\mathsf{PUF}}, \mathcal{P}_S, \mathsf{C}_{\mathsf{puflideal}}, \mathsf{R}_{\mathsf{puflideal}})$ to $\mathcal{F}_{\mathsf{PUF}}$.
2. Receiver $\mathsf{R}_{\mathsf{puflideal}}$: obtain p'_S from the committer and $(\mathsf{handover}_{\mathsf{PUF}}, \mathcal{P}_S, \mathsf{C}_{\mathsf{puflideal}})$ from $\mathcal{F}_{\mathsf{PUF}}$. Pick random string $r \in \{0,1\}^\ell$ and send it to the committer.
3. Committer $\mathsf{C}_{\mathsf{puflideal}}$: send $\mathbf{c} = st_S \oplus (r \wedge b^\ell)$ to $\mathsf{R}_{\mathsf{puflideal}}$.

Decommitment Phase

1. Committer $\mathsf{C}_{\mathsf{puflideal}}$: send (b, q) to $\mathsf{R}_{\mathsf{puflideal}}$.
2. Receiver $\mathsf{R}_{\mathsf{puflideal}}$: obtain (b', q') from the committer. Send $(\mathsf{eval}_{\mathsf{PUF}}, \mathcal{P}_S, \mathsf{R}_{\mathsf{puflideal}}, q')$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain $(\mathsf{response}_{\mathsf{PUF}}, \mathcal{P}_S, q', a')$. Compute $st'_S \leftarrow \mathsf{FuzRep}(a', p'_S)$. If $b = 0$, check if $st'_S = \mathbf{c}$. Else, check $st'_S = \mathbf{c} \oplus r$. If the check passes, accept and output b, else output reject.

Figure 3.3: $\mathsf{IdealComPuf}$*: Ideal Commitments in the* $\mathcal{F}_{\mathsf{PUF}}$ *model.*

Theorem 5. *Protocol* $\mathsf{IdealComPuf}$ *is an ideal commitment scheme in the* $\mathcal{F}_{\mathsf{PUF}}$ *model.*

Proof. Completeness of protocol $(\mathsf{C}_{\mathsf{puflideal}}, \mathsf{R}_{\mathsf{puflideal}})$ follows from response consistency. We focus on hiding and binding properties.

Lemma 12 (Hiding)**.** *For any malicious receiver* $\mathsf{R}^*_{\mathsf{puflideal}}$*, the statistical difference between the ensembles*

$$\{\, \mathsf{view}_{\mathsf{R}^*}(\mathsf{c}(\mathtt{com}, 0), \mathsf{R}^*(\mathtt{recv}))(1^n) \,\}_{n\in\mathbb{N}} \text{ and } \{\, \mathsf{view}_{\mathsf{R}^*}(\mathsf{c}(\mathtt{com}, 1), \mathsf{R}^*(\mathtt{recv}))(1^n)) \,\}_{n\in\mathbb{N}}$$

is negligible in n*.*

Proof. Let $\mathcal{Q}$ be the set of queries that the receiver R^* makes to the committer's PUF sid and let q be the query made by the committer before sending the PUF sid. First

consider the case that there exists $q' \in \mathcal{Q}$ such that $\mathsf{dis}(q', q) < d_{\mathsf{min}}$. As the receiver is polynomially bounded, the number of queries in $\mathcal{Q}$ is a polynomial, say $p(n)$. The total number of queries within a distance d_{min} of queries in $\mathcal{Q}$ can be bounded by $p(n)n^{d_{\mathsf{min}}(n)}$, which is a negligible fraction of 2^n. Thus, this event happens with negligible probability.

Now consider the case that $q \notin \mathcal{Q}$. By the extraction independence property, st is statistically close to the uniform distribution, U_ℓ. As, for any string r, the distributions U_ℓ and $r \oplus U_\ell$ are identical, thus it follows from transitivity that the distributions st and $st \oplus r$ are statistically close.

□

We now turn to the binding property. The proof follows a similar path as in the proof of statistical binding in Naor's commitment [76]. Informally[3], Naor's argument counts the number of 'bad' strings in the range of the PRG. These are the strings r in the range of a PRG $G(\cdot)$ for which there exist two seeds s_0, s_1 such that $r = G(s_0) \oplus G(s_1)$. For these strings r, equivocation is possible. But because of the expansion property of PRG, the number of bad strings is small. Similarly, in our proof of binding, we use the fact that we have set the parameters of the PUF family and fuzzy extractor such that $\ell = 3n$. We use the same arguments to define 'bad' strings and show that because of expansion, their number is bounded. Care has to be taken to handle the fact that the output of the PUF is noisy.

Lemma 13 (Binding)**.** *For any malicious committer $\mathsf{C}^*_{\mathsf{pufIdeal}}$, the probability that it wins in the binding game of Definition 1 is negligible in n.*

Proof. Recall that we have chosen the parameters of the PUF family and fuzzy extractor such that $\ell = 3n$. We can think of the adversary choosing the malicious PUF as picking a set of distributions $\mathcal{D}_{q_1}, \ldots, \mathcal{D}_{q_N}$, where $N = 2^n$. For a fixed p, call a string $st \in \{0,1\}^\ell$ "heavy" if there exists query q such that $\mathsf{Pr}\left[\, \mathsf{FuzRep}(p, \mathcal{D}_q) = st \,\right] \geq 2^{-\log^2(n)}$.

Now we will show that the probability of the adversary breaking the binding is negligible. For a fixed first message of the malicious committer, call a string $r \in \{0,1\}^\ell$ 'bad' if the probability that the adversary breaks binding on receiving r in the second step of the protocol is at least $2^{-2\log^2(n)}$. For this to happen, it must be the case that there exist heavy strings st_0 and st_1 such that $r = st_0 \oplus st_1$. Thus, to bound the number of bad strings in $\{0,1\}^\ell$, we simply need to bound the number of pairs of heavy strings. By the definition of heavy string, each query can produce at most $2^{\log^2(n)}$ heavy strings for one PUF. As the total number of queries is 2^n, the total number of pairs of heavy strings is bounded by $2^{2(n+\log^2(n))}$, which is a negligible fraction of 2^{3n}.

□

□

[3]The following assumes familiarity with Naor's commitment scheme [76].

Ideal *Extractable* Commitment in the $\mathcal{F}_{\mathsf{PUF}}$ Model. We transform $\mathsf{IdealComPuf}$ into a *straight-line extractable* commitment using the following technique. We introduce a new PUF $\mathsf{PUF_R}$, sent by the receiver to the committer, at the beginning of the protocol. Then we force the committer to query the PUF $\mathsf{PUF_R}$ with the opening of the commitment computed running $\mathsf{IdealComPuf}$. An opening of protocol $\mathsf{IdealComPuf}$ is the value a [4] In this way, the extractor, having access to the interface of $\mathcal{F}_{\mathsf{PUF}}$, intercepts the queries made by the committer, and thus extracts the opening. Note that extractability must hold against a malicious committer, in which case the token $\mathsf{PUF_R}$ sent by the receiver is honest, therefore the extractor is allowed to intercept such queries. The idea is that, from the transcript of the commitment (i.e., the value $\mathbf{c} = a \oplus (r \wedge b)$) and the queries made to $\mathsf{PUF_R}$, (the value σ_S) the bit committed if fully determined [5].

How can we force the committer to query $\mathsf{PUF_R}$ with the correct opening? We require that it commits to the answer σ_R obtained by $\mathsf{PUF_R}$, using again protocol $\mathsf{IdealComPuf}$. Why the committer cannot send directly the answer σ_R? Because σ_R could be the output of a malicious PUF, and leak information about the query made by the committer.

Thus, in the commitment phase, committer runs two instances of $\mathsf{IdealComPuf}$. One instance, that we call ComBit, is run to commit to the secret bit b. The other instance, that we call $\mathsf{ComResp}$, is run to commit to the response of PUF $\mathsf{PUF_R}$, queried with the opening of ComBit. In the decommitment phase, the receiver gets $\mathsf{PUF_R}$ back, along with the openings of the bit and the PUF-response. Then it queries $\mathsf{PUF_R}$ with the opening of ComBit, and checks if the response is consistent with the string committed in $\mathsf{ComResp}$.

Due to the unpredictability of PUFs, the committer cannot guess the output of $\mathsf{PUF_R}$ on the string σ_S without querying it. Due to the statistically binding of $\mathsf{IdealComPuf}$, the committer cannot postpone querying the PUF in the decommitment phase. Thus, if the committer will provide a valid decommitment, the extractor would have observed the opening already in the commitment phase with all but negligible probability.

However, there is one caveat. The unpredictability of PUFs is guaranteed only for queries that are sufficiently apart from each other. Which means that, given a challenge/response pair (c, r), the response on any strings c' that is "close" in hamming distance to c ("close" means that $\mathsf{dis_{ham}}(c, c') \leq d_{\mathsf{min}}$), could be predictable.

Consequently, a malicious committer could query the PUF with a string that is only "close" to the opening. Then, given the answer to such a query, she could predict the answer to the actual opening, *without* querying the PUF. In this case, the extractor cannot determine which is the opening, since it cannot try all possible strings that are "close" to queries made by the malicious committer. Thus the extraction fails. At the

[4] In the actual implementation we require the committer to query $\mathsf{PUF_R}$ with st_S where $(st_S, p_S) \leftarrow \mathsf{FuzGen}^1(a)$.

[5] As we shall discuss in the security proof, a malicious sender can always compute $\mathbf{c}^*$ so that it admits two valid openings (i.e., compute y_0, y_1 such that $r = y_0 \oplus y_1$ and set $\mathbf{c}^* = y_0$) , and query $\mathsf{PUF_R}$ with both openings (thus confusing the extractor). However, due to the binding of $\mathsf{IdealComPuf}$, $\mathcal{A}$ will not be able to provide an accepting decommitment for such $\mathbf{c}^*$. Thus extractability is not violated. (Straight-line Extractability in $\mathcal{F}_{\mathsf{aux}}$ model, is violated when the extractor outputs $\perp$, while the adversary provides an accepting decommitment).

same time, the malicious committer did not violate the unpredictability property of PUFs, since it predicted a value that is "too close" to the one already observed.

We overcome this problem by using Error Correcting Codes, in short ECC (see Definition 10). The property of an ECC with distance parameter dis, is that any pair of strings having hamming distance dis, decodes to a unique string. Therefore, we modify the previous approach asking the committer to query PUF_R with the *encoding* of the opening, i.e., $\mathsf{Encode}(\sigma_S)$. In this way, all queries that are "too close" in hamming distance, decode to the same opening, and the previous attack is defeated.

Informally, hiding and biding follow from properties of IdealComPuf. Indeed, protocol ComExtPuf, basically consists in running two instances of IdealComPuf in parallel. Extractability follows from the statistically biding of IdealComPuf, the unpredictability of PUF_R and the Minimum Distance Property of ECC.

The formal description of the above protocol, that we denote by $\mathsf{ComExtPuf} = (\mathsf{C}_\mathsf{pufExt}, \mathsf{R}_\mathsf{pufExt})$, is shown in Fig. 3.4.

Replacement of the honest PUF. In the decommitment phase, the committer sends back PUF_R to the receiver. The receiver checks the validity of the decommitment by querying PUF_R with the decommitment data. A malicious committer, could replace PUF_R with another PUF, in which case the extractability property is not achieved anymore. This attack can be easily overcome by assuming that, before giving its own PUF PUF_R away, the receiver queries it with a secret random challenge, and stores the response. Then, when PUF_R is sent back, the receiver checks its authenticity by querying PUF_R on the same challenge and matching the response obtained with the one stored.

On round complexity of ComExtPuf. For simplicity in Fig. 3.4 we describe the interaction between $\mathsf{C}_\mathsf{pufExt}$ and $\mathsf{R}_\mathsf{pufExt}$ using several rounds. However, we stress out that, the exchange of the PUF can be done once at the beginning of the protocol, and that except from the PUF transfer, the commitment phase requires only three rounds. The decommitment is non-interactive, and require another PUF transfer.

Theorem 6. *If* IdealComPuf *is an Ideal Commitment in the* $\mathcal{F}_\mathsf{PUF}$*-model, then* ComExtPuf *is an Ideal* Extractable *Commitment in the* $\mathcal{F}_\mathsf{PUF}$ *model.*

Proof. **Completeness.** Completeness follows from completeness of IdealComPuf, from the response consistency property of PUF and fuzzy extractors and the correct decoding property of Error Correcting Codes.

Hiding. The commitment phase of protocol ComExtPuf basically consists in the parallel execution of two instances of IdealComPuf. In the first instance, that we call ComBit, $\mathsf{C}_\mathsf{pufExt}$ commits to its secret bit b, in the other instance, that we call ComResp, it commits to some value received from the (possibly malicious) PUF $\mathcal{P}_R^*$ [6]. Although $\mathcal{P}_R^*$ could compute the response *adaptively* on the query observed, thus revealing information about the opening (recall that the query corresponds to the opening of ComBit), such information cannot reach $\mathcal{A}$ since the response is

[6] Recall that, to create a malicious PUF, the malicious receiver $\mathcal{A}$ sends $(\mathsf{init}_\mathsf{PUF}, \mathtt{mal}, \mathsf{PUF}_\mathsf{R}, \mathsf{R}_\mathsf{pufExt})$ to $\mathcal{F}_\mathsf{PUF}$

committed using $\mathsf{IdealComPuf}$. Furthermore in case $\mathcal{P}_R^*$ aborts, $\mathsf{C}_{\mathsf{pufExt}}$ continues the protocol, committing to the string 0, in fact, ruling out selective abort attacks.

Formally, the hiding proof goes by hybrids:

H_0 : In this experiment the committer honestly commits to the bit 0. Namely, it runs ComBit to commit to 0, then in queries the possibly malicious PUF $\mathcal{P}_R^*$ with the opening of ComBit. Finally it commits to the answer received from $\mathcal{P}_R^*$ running protocol $\mathsf{ComResp}$ (if $\mathcal{P}_R^*$ aborts, the committer commits to the zero string).

H_1 : In this experiment the committer runs ComBit as commitment of 0 and $\mathsf{ComResp}$ as commitment of the string 0^ℓ, instead of the actual opening of ComBit. Due to the hiding of $\mathsf{IdealComPuf}$, H_0 and H_1 are statistically close.

H_2 : In this experiment the commitment runs ComBit as commitment of 1 and $\mathsf{ComResp}$ still as commitment of 0^ℓ. Due to the hiding of $\mathsf{IdealComPuf}$, H_1 and H_2 are statistically close.

H_3 : In this experiment the committer queries the possibly malicious PUF $\mathcal{P}_R^*$ with the opening of ComBit and commits to the answer (if any) running $\mathsf{ComResp}$. If $\mathcal{P}_R^*$ aborts, the committer commits to the zero string. Due to the hiding of $\mathsf{IdealComPuf}$, H_2 and H_3 are statistically close. In this experiment the committer is honestly committing to the bit 1. This completes the proof.

Binding. Binding follows straight-forwardly from the binding property of $\mathsf{IdealComPuf}$.

Extractability. We show a straight-line PPT extractor E that having interface access to $\mathcal{F}_{\mathsf{PUF}}$ satisfies the properties of Definition 18. The extractor is formally described in Fig. 3.5. $\mathcal{A}$ denotes the malicious sender.

Extractor E satisfies the following properties.

E **runs in polynomial time.** E follows the procedure of the honest receiver, which is polynomial. In the extraction phase E runs algorithm dec for at most polynomially many queries. Due to the efficiency property of ECC this operation also requires polynomial time.

Simulation. The extractor E follows the procedure of the honest receiver $\mathsf{R}_{\mathsf{pufExt}}$, and additionally it collects the queries made by $\mathcal{A}$ to $\mathsf{PUF}_{\mathsf{R}}$. Therefore the view of $\mathcal{A}$ interacting with E is identical to the view of $\mathcal{A}$ interacting with $\mathsf{R}_{\mathsf{pufExt}}$.

Extraction. We have to prove that, when E outputs $\perp$, probability that $\mathcal{A}$ provides an accepting decommitment is negligible. First, recall that E outputs $\perp$ in two cases. Case 1) there exists a pair of queries x_0, x_1 that are both valid openings of $\mathbf{c}$. Case 2) there exists no query decoding to a valid opening of $\mathbf{c}$.

- Case 1. Note that, $\mathcal{A}$ can always compute x_0, x_1 such that $r = \mathsf{dec}(x_0) \oplus \mathsf{dec}(x_1)$ and compute $\mathbf{c} = \mathsf{dec}(x_0)$. We argue that, if $\mathcal{A}$ computes $\mathbf{c}$ in such

a way, then probability that $\mathcal{A}$ can provide an accepting decommitment for $\mathbf{c}$ is negligible. This follows directly from the binding of $\mathsf{IdealComPuf}$.

- Case 2. Towards a contradiction, assume that $\mathcal{A}$ does not query the PUF with any valid opening, but in the decommitment phase, $\mathcal{A}$ still provides an accepting decommitment. An accepting decommitment in $\mathsf{ComExtPuf}$ consists of the decommitments of ComBit and $\mathsf{ComResp}$. Namely, the bit b, along with the value st_S such that $\mathbf{c} = st_S \oplus (r \wedge b)$, and the string $(st_R||p_R)$ (for simplicity we are omitting the remaining decommitment data).
  Since the decommitment is accepting it holds that st_R is the answer of the **honest** PUF $\mathsf{PUF_R}$ on the query $\mathsf{Encode}(st_S)$(more precisely $st_R = \mathsf{FuzRep}(\sigma_R, p_R)$ where σ_R is the actual answer of $\mathsf{PUF_R}$ on input $\mathsf{Encode}(st_S)$).
  By hypothesis no queries received by $\mathsf{PUF_R}$ in the commitment phase decoded to st_S. Thus one of these two cases has happened:
  1. $\mathcal{A}$ has correctly computed $\mathsf{PUF_R}$'s responds σ_R without querying $\mathsf{PUF_R}$. In this case $\mathcal{A}$ breaks unpredictability of the honest PUF $\mathsf{PUF_R}$.
     Indeed, due to the Minimum Distance property of ECC, we have that all the valid codewords are at d_{min} hamming distance from each other. Thus, the only way for $\mathcal{A}$ to obtain a response for an encoding of st_S that was not inferred by E, is that such encoding is d_{min} apart from any challenge observed by E. Predicting the PUF-response of a challenge that is so far from the previously queried challenges, corresponds to break the unpredictability of the PUF.
  2. $\mathcal{A}$ queries $\mathsf{PUF_R}$ only in the decommitment phase. Then she opens the commitment of the response, $\mathsf{ComResp}$, accordingly. Due to the statistically binding property of $\mathsf{IdealComPuf}$, this case happens with negligible probability.

Binding. Here we have to prove that if E extracts bit b, probability that $\mathcal{A}$ decommits to bit $\bar{b}$ is negligible. This basically follows from the biding of the sub-protocol $\mathsf{IdealComPuf}$.

□

Protocol ComExtPuf

$\mathsf{ECC} = (\mathsf{Encode}, \mathsf{Decode})$ is a $(N, L, d^1_{\mathsf{min}})$ error correcting code. $\mathcal{F}_{\mathsf{PUF}}$ is parameterized with a PUF family $\mathcal{P}$=$(rg^1, d^1_{\mathsf{noise}}, d^1_{\mathsf{min}}, \tilde{m}^1)$, with challenge size L. $(\mathsf{FuzGen}^1, \mathsf{FuzRep}^1)$ is a $(\tilde{m}^1, \ell^1, t^1, \epsilon^1)$-fuzzy extractor of appropriate matching parameters. Protocol $\mathsf{IdealComPuf} = (\mathsf{C}_{\mathsf{pufIdeal}}, \mathsf{R}_{\mathsf{pufIdeal}})$ (depicted in Fig. 3.3) is run as sub-routine. $\mathcal{P}_S, \mathsf{PUF}_\mathsf{R}$ denote (sid of) the PUF created by the committer and the receiver, respectively.

Committer's Input: $b \in \{0, 1\}$.

Commitmen Phase

1. Receiver $\mathsf{R}_{\mathsf{pufExt}}$: create a PUF sending $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{PUF}_\mathsf{R}{}^a, \mathtt{normal}, \mathsf{R}_{\mathsf{pufExt}})$ to $\mathcal{F}_{\mathsf{PUF}}$ and then handover it to $\mathsf{C}_{\mathsf{pufExt}}$, sending $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{PUF}_\mathsf{R}, \mathsf{R}_{\mathsf{pufExt}}, \mathsf{C}_{\mathsf{pufExt}})$ to $\mathcal{F}_{\mathsf{PUF}}$.

2. **Commitment of the Secret Bit:** ComBit**.**

   $\mathsf{C}_{\mathsf{pufExt}} \Leftrightarrow \mathsf{R}_{\mathsf{pufExt}}$: run $\langle \mathsf{C}_{\mathsf{pufIdeal}}(\mathtt{com}, b), \mathsf{R}_{\mathsf{pufIdeal}}(\mathtt{com}) \rangle$ so that $\mathsf{C}_{\mathsf{pufExt}}$ commits to bit b.

   Let $(st_S, p_S) \leftarrow \mathsf{FuzGen}^1(a)$ the value obtained by $\mathsf{C}_{\mathsf{pufExt}}$, after applying the fuzzy extractor to the answer obtained from $\mathcal{P}_S$ in protocol ComBit.

3. Committer $\mathsf{C}_{\mathsf{pufExt}}$: Send $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{PUF}_\mathsf{R}, \mathsf{C}_{\mathsf{pufExt}}, \mathsf{Encode}(st_S))$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain response $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{PUF}_\mathsf{R}, \mathsf{Encode}(st_S), \sigma_R)$. If $\sigma_R = \perp$ (i.e., PUF PUF_R aborts), set $\sigma_R \leftarrow 0$. Compute $(st_R, p_R) \leftarrow \mathsf{FuzGen}^1(\sigma_R)$.

4. **Commitment of** PUF_R**'s Response:** ComResp**.**

   $\mathsf{C}_{\mathsf{pufExt}} \Leftrightarrow \mathsf{R}_{\mathsf{pufExt}}$: run $\langle \mathsf{C}_{\mathsf{pufIdeal}}(\mathtt{com}, st_R||p_R), \mathsf{R}_{\mathsf{pufIdeal}}(\mathtt{com}) \rangle$ so that $\mathsf{C}_{\mathsf{pufExt}}$ commits to the string $st_R||p_R$.

Decommitment Phase

1. $\mathsf{C}_{\mathsf{pufExt}} \Leftrightarrow \mathsf{R}_{\mathsf{pufExt}}$:
   run $\langle \mathsf{C}_{\mathsf{pufIdeal}}(\mathtt{open}, b), \mathsf{R}_{\mathsf{pufIdeal}}(\mathtt{open}) \rangle$ and $\langle \mathsf{C}_{\mathsf{pufIdeal}}(\mathtt{open}, st_R||p_R), \mathsf{R}_{\mathsf{pufIdeal}}(\mathtt{open}) \rangle$.

2. Committer $\mathsf{C}_{\mathsf{pufExt}}$: handover PUF to $\mathsf{R}_{\mathsf{pufExt}}$ by sending $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{PUF}_\mathsf{R}, \mathsf{C}_{\mathsf{pufIdeal}}\mathsf{R}_{\mathsf{pufExt}})$ to $\mathcal{F}_{\mathsf{PUF}}$.

3. Receiver $\mathsf{R}_{\mathsf{pufExt}}$: If both decommitment are successfully completed, then $\mathsf{R}_{\mathsf{pufExt}}$ gets the bit b' along with the opening st'_S for ComBit and string $st'_R||p'_R$ for ComResp.
   Check validity of st'_R: send $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{PUF}_\mathsf{R}, \mathsf{R}_{\mathsf{pufExt}}, \mathsf{Encode}(st'_S))$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain σ'_R. Compute $st''_R \leftarrow \mathsf{FuzRep}^1(\sigma'_R, p'_R)$. If $st''_R = st'_R$, then accept and output b. Else, reject.

[a] PUF_R corresponds to the id of the PUF.

Figure 3.4: ComExtPuf*: Ideal* **Extractable** *Commitment in the* $\mathcal{F}_{\mathsf{PUF}}$ *model.*

Extractor E
E creates PUF $\mathsf{PUF_R}$ sending $(\mathsf{init_{PUF}}, \mathtt{normal}, \mathsf{PUF_R}, \mathsf{R_{pufExt}})$ to $\mathcal{F}_{\mathsf{PUF}}$. E handovers the PUF to $\mathcal{A}$, sending $(\mathsf{handover_{PUF}}, \mathsf{PUF_R}, \mathsf{R_{pufExt}}, \mathcal{A})$ to $\mathcal{F}_{\mathsf{PUF}}$. Queries made by $\mathcal{A}$ to $\mathsf{PUF_R}$ are intercepted by E, stored in the variable $\mathcal{Q}$, and then forwarded to $\mathcal{F}_{\mathsf{PUF}}$. The answers received by $\mathcal{F}_{\mathsf{PUF}}$ are then forwarded to $\mathcal{A}$.
Commitment Phase:
E honestly follows the procedure of $\mathsf{R_{pufExt}}$. If the commitment phase is accepting, E proceeds to extraction phase. Else, it halts. Let $(r, \mathbf{c})$ be the transcript of ComBit.
Extraction Phase:
- If there exists a query $x \in \mathcal{Q}$ such that $\mathbf{c} = \mathsf{dec}(x)$ then output 0.
- If there exists a query $x \in \mathcal{Q}$ such that $\mathbf{c} = \mathsf{dec}(x) \oplus r$ then ouput 1.
- Case 1) If there exist queries $x_0, x_1 \in \mathcal{Q}$ s.t. $\mathbf{c} = \mathsf{dec}(x_0)$ AND $\mathbf{c} = \mathsf{dec}(x_1) \oplus r$ then output $\perp$.
- Case 2) If there exist no query in $\mathcal{Q}$ that decodes to a valid opening of $\mathbf{c}$, output $\perp$.

Figure 3.5: *E: Extractor associated to* $\mathsf{ComExtPuf}$.

3.4 Ideal Extractable Commitments from Stateless Tokens

In this section we show how to construct ideal extractable commitments from stateless tokens. We first construct an ideal commitment scheme. Then, we use it as building block for constructing an ideal *extractable* commitment.

Ideal Commitment Scheme in the $\mathcal{F}_{\mathsf{wrap}}$ Model. Similarly to the construction with PUFs, we implement Naor's commitment scheme by replacing the PRG with a stateless token. Note that Naor's commitment is already statistically binding, therefore the token is only used to obtain also statistically hiding. In the construction with PUFs, the PRG is replaced with a PUF that is inherently unpredictable. Thus, even after observing a polynomial number of challenge/response pairs, a malicious receiver cannot predict the output of the PUF on any new (sufficiently far apart) challenge. In this case, hiding breaks only if the receiver guesses the challenge used by the committer, which happens only with negligible probability. Hence, hiding holds unconditionally.

Now, we want to achieve statistically hiding using *stateless* token. The problem here is that we do not have unpredictability by default (as it happens with PUFs). Thus, we have to program the stateless token with a function that is, somehow, unconditionally unpredictable. Clearly, we cannot construct a token that implements a PRG. Indeed, after observing a few pairs of input/output, an unbounded receiver can extract the seed, compute all possible outputs, and break hiding. We follow that same idea as [53] and we use a point function. A point function f is a function that outputs always zero, except in a particular point x, in which it outputs a value y. Formally, $f : \{0,1\}^n \to \{0,1\}^m$ such that $f(x) = y$ and $f(x') = 0$, for all $x' \neq x$.

Thus, we adapt Naor's commitment scheme as follows. The committer picks a n-bit string x and a $3n$-bit string y and creates a stateless token that on input x outputs y, while it outputs 0 on any other input. The stateless token is sent to the receiver at the beginning of the protocol. After having obtained the token, the receiver sends then Naor's first message, i.e., the random value r, to the committer. The committer commits to the bit b by sending $\mathbf{c} = y \oplus (r \wedge b)$. In the decommitment phase, the committer sends x, y, b. The receiver queries the token with x and obtain a string y'. If $y = y'$ the receiver accepts iff $\mathbf{c} = y' \oplus (r \wedge b)$.

Statistically binding follows from the same arguments of Naor's scheme. The token is sent away before committer can see r. Thus, since x is only n bits, information theoretically the committer cannot instruct a malicious token to output y' adaptively on x. Thus, for any malicious possibly *stateful* token, binding is preserved.

Statistically hiding holds due to the fact that x is secret. A malicious receiver can query the token with any polynomial number of values x'. But, whp she will miss x, and thus she will obtain always 0. Such an answer does not help her to predict y. The only way to obtain y is to predict x. This happens with probability 2^{-n}.

The above protocol is denoted by $\mathsf{IdealTok}$ and is formally described in Fig. 3.6. We stress that, this is the first construction of unconditional commitment scheme that is secure even against malicious *stateful* tokens. The previous construction of unconditional

commitment scheme of [53] is secure as long as the malicious token is stateless (i.e., it assumes that the adversary cannot create stateful tokens). Furthermore, our constructions requires only one stateless token, while construction of [53] requires a number of tokens that depends on the security parameter.

String Commitment reusing the same token. To commit to a $\ell = \mathtt{poly}(n)$ bit string *reusing* the same stateless tokens is sufficient to embed ℓ pairs $(x_1, y_1),\ldots,(x_\ell, y_\ell)$ in $\mathcal{T}_{\mathsf{C}}$. Then, execute protocol $\mathsf{IdealTok}$ for each bit of the string in parallel. The receiver accepts the string iff all bit commitments are accepting. Hiding still holds since probability that a malicious receiver guesses one of the challenges is $\ell/2^n$.

Protocol $\mathsf{IdealTok}$
Committer's Input: $b \in \{0,1\}$.

Commitmen Phase

1. Committer $\mathsf{C}_{\mathsf{IdealTok}}$: pick $x \xleftarrow{\$} \{0,1\}^n$, $y \xleftarrow{\$} \{0,1\}^{3n}$. Create token $\mathcal{T}_{\mathsf{C}}$ implementing the point function $f(x) = y$; $f(x') = 0$ for all $x' \neq x$. Send $(\mathsf{create}, \mathsf{sid}, \mathsf{C}_{\mathsf{IdealTok}}, \mathsf{R}_{\mathsf{IdealTok}}, \mathcal{T}_{\mathsf{C}})$ to $\mathcal{F}_{\mathsf{wrap}}$.
2. Receiver $\mathsf{R}_{\mathsf{IdealTok}}$: pick $r \xleftarrow{\$} \{0,1\}^{3n}$. Send r to $\mathsf{C}_{\mathsf{IdealTok}}$.
3. Committer $\mathsf{C}_{\mathsf{IdealTok}}$: Send $\mathbf{c} = y \oplus (r \wedge b^{3n})$ to $\mathsf{R}_{\mathsf{IdealTok}}$.

Decommitment Phase

1. Committer $\mathsf{C}_{\mathsf{IdealTok}}$: send (b, x) to $\mathsf{R}_{\mathsf{IdealTok}}$.
2. Receiver $\mathsf{R}_{\mathsf{IdealTok}}$: send $(\mathsf{run}, \mathsf{sid}, \mathsf{R}_{\mathsf{IdealTok}}, \mathcal{T}_{\mathsf{C}}, x)$ and obtain y. If $b = 0$, check that $\mathbf{c} = y$. Else, check that $y = \mathbf{c} \oplus r$. If the check passes, accept and output b, else reject.

Figure 3.6: $\mathsf{IdealTok}$*: Ideal Commitments in the* $\mathcal{F}_{\mathsf{wrap}}$ *model.*

String Commitment reusing the same token. To commit to a $\ell = \mathtt{poly}(n)$ bit string *reusing* the same stateless tokens is sufficient to embed ℓ pairs $(x_1, y_1),\ldots,(x_\ell, y_\ell)$ in $\mathcal{T}_{\mathsf{C}}$. Then, execute protocol $\mathsf{IdealTok}$ for each bit of the string in parallel. The receiver accepts the string iff all bit commitments are accepting. Hiding still holds since probability that a malicious receiver guesses one of the challenges is $\ell/2^n$.

Ideal Extractable Commitment in the $\mathcal{F}_{\mathsf{wrap}}$ Model. To achieve extractability we again augment the ideal commitment scheme $\mathsf{IdealTok}$ with three simple steps. First, make the receiver send a token $\mathcal{T}_{\mathsf{R}}$ to the committer. Second, make the committer to first commit to the secret bit (using $\mathsf{IdealTok}$), then query token $\mathcal{T}_{\mathsf{R}}$ with the *opening* of such commitment (namely, value y). Third, the committer commits to the answer received from $\mathcal{T}_{\mathsf{R}}$. In the decommitment phase, the receiver has to cross check the validity of

two commitments (commitment of the bits and commitment of the answer) and the consistency of the answer with its own token. Note that with tokens there is no need for the committer to send the token back to the receiver, since the function embedded in the token is well known to the creator.

However, with stateless tokens, achieving extractability is more complicated. Indeed, which function should $\mathcal{T}_{\mathsf{R}}$ run, that will force the committer to query it with the correct opening? Let us discuss some wrong solution, to then converge to the correct one.

Consider the function that takes as input a string y (the opening of ComBit) and outputs $\mathsf{Mac}(k, y)$, for some secret key k. Such function does not guarantee extractability. A malicious committer, can query the token on two random strings y_1, y_2 (the token is stateless) and extract the MAC key. Later, the adversary can secretly compute the MAC on the actual opening y, without using the token. Thus, she will be able to provide a valid decommitment, while the extractor fails. Note that, this case is ruled out when using PUFs. The reason is that, even after many queries, the adversary is not able to compute the answer of the PUF on a new string y by herself.

Consider the function that takes as input a commitment's transcript $(r, \mathbf{c})$ (of protocol $\mathsf{IdealTok}$) and the opening y. It checks that y is a correct opening of $\mathbf{c}$, and if so, it outputs $\mathsf{Mac}(k, y)$. This function is still not sufficient for obtaining extraction. A malicious committer can query the token with arbitrary pairs (commitment, decommitment) that do not correspond to the actual commitment $\mathbf{c}$ sent to the receiver. Thus we are in the previous case again.

The right function to embed in the stateless token, is the following. The function, parameterized by two independent MAC keys $k_{\mathsf{rec}}, k_{\mathsf{tok}}$, takes as input a commitment's transcript $(r, \mathbf{c})$, a MAC σ_{rec} (value σ_{rec} is computed by the receiver, i.e., the creator of the token) and an opening y. The function checks that y is a valid opening of $(r, \mathbf{c})$, and that σ_{rec} is a valid MAC computed on $(r, \mathbf{c})$ (i.e., $\sigma_{\mathsf{rec}} = \mathsf{Mac}(k_{\mathsf{rec}}, r||\mathbf{c})$). If both checks are successful, the function outputs the MAC computed on the opening y. Namely, it outputs $\sigma_{\mathsf{tok}} = \mathsf{Mac}(k_{\mathsf{tok}}, y)$. Due to the unforgeability of the MAC, and the statistically binding property of the commitment scheme $\mathsf{IdealTok}$, a malicious committer can successfully obtain the answer to exactly one query. Note that, a malicious committer, can perform the following attack. Once it receives the string r from the receiver, it picks strings y_0 and y_1 such that $r = y_0 \oplus y_1$ and sends the commitment $\mathbf{c} = y_0$ to the receiver, obtaining the MAC of $\mathbf{c}$. With the commitment so computed, and the MAC, it can query token $\mathcal{T}_{\mathsf{R}}$ twice with each valid opening. In this case, the committer can extract the MAC key, and at the same time baffling the extractor that observes two valid openings. The observation here is that, due to the binding of $\mathsf{IdealTok}$, for a commitment $\mathbf{c}$ computed in such a way, the malicious committer will not be able, in the decommitment phase, to provide a valid opening. (The reason is that, whp, she cannot instruct its token to output neither y_0 or y_1). Thus, the extractor fails and outputs $\perp$, but at the same time the decommitment will not be accepting. Thus extractability is not violated.

As final step, the committer commits to the answer σ_{tok}, using the scheme $\mathsf{IdealTok}$. (If the token of the receiver aborts, the committer sets σ_{tok} to the zero string). In the decommitment phase, the receiver, first checks the validity of both commitments

(commitment of the bit, commitment of the answer σ_{tok}). Then, given the opening of the bit, it checks that σ_{tok} is a valid MAC computed under key k_{tok} on such opening.

Binding follows from the binding of IdealTok and the unforgeability of MAC. Hiding still follows from the hiding of IdealTok. Indeed, the answer of $\mathcal{T}_{\mathsf{R}}$ sent by the malicious receiver, is not forwarded to the receiver, but is committed using the ideal commitment IdealTok. Furthermore, if $\mathcal{T}_{\mathsf{R}}$ selective aborts, the committer does not halt, but it continues committing to the zero-string. The receiver will get the answer in clear, only in the decommitment phase, when the bit has been already revealed. The formal description of the above protocol, that we denote by ExtTok, is shown in Fig. 3.7.

Extractable String Commitment reusing the same token. To achieve extractability for a ℓ-bit string, reusing the same token $\mathcal{T}_{\mathsf{R}}$ is sufficient to load $\mathcal{T}_{\mathsf{R}}$ with ℓ pairs of independently chosen MAC keys $(k^1_{\mathsf{rec}}, k^1_{\mathsf{tok}}), \ldots, (k^\ell_{\mathsf{rec}}, k^\ell_{\mathsf{tok}})$. Then run ℓ executions of ExtTok in parallel. The receiver accepts the string, iff the opening of all ℓ bits are accepting. Extractability holds due to the fact that for each execution of ExtTok, a new MAC key is used.

Theorem 7. *Protocol* ExtTok *is an ideal* extractable *commitment in the* $\mathcal{F}_{\mathsf{wrap}}$ *model.*

Proof. First, we prove that IdealTok is an ideal commitment scheme in the $\mathcal{F}_{\mathsf{wrap}}$ model.

Theorem 8. *Protocol* IdealTok *is an ideal commitment scheme in the* $\mathcal{F}_{\mathsf{wrap}}$ *model.*

Proof. **Completeness.** By inspection.

Hiding. Hiding breaks if a malicious receiver $\mathcal{A}$ is able to compute y, in the commitment phase. Recall that values x, y embedded into the stateless token $\mathcal{T}_{\mathsf{c}}$ are chosen uniformly at random. Furthermore, $\mathcal{T}_{\mathsf{c}}$ responds only on input x. Since $\mathcal{A}$ can make only polynomial number of queries to $\mathcal{T}_{\mathsf{c}}$, it can get y only if she guesses x. This happens with negligible probability only.

Binding. The proof of binding can be adapted from the proof of protocol IdealCom. It is sufficient to observe that a malicious PUF can be a malicious token.

□

We are now ready to prove Theorem 7.

Proof. **Completeness.** Due to the completeness of the one-time unconditional MAC and the completeness of the sub-protocol IdealTok.
Hiding. Follows directly from the hiding property of protocol IdealTok. The formal argument is similar to the one provided in the hiding proof of Section 3.3, and is therefore omitted.
Binding. Follows directly from the binding property of protocol IdealTok.
Extractability. Extractability roughly follows from the binding of IdealTok and the unconditional one-time unforgeability of MAC. Details follow.

We show a straight-line PPT extractor E that having interface access to $\mathcal{F}_{\mathsf{wrap}}$ satisfies the properties of Definition 18. The extractor is formally described in Fig. 3.8. $\mathcal{A}$ denotes the malicious sender.

Protocol ExtTok

$(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ is a one-time unconditional MAC. Protocol $\mathsf{IdealTok} = (\mathsf{C}_{\mathsf{IdealTok}}, \mathsf{R}_{\mathsf{IdealTok}})$ is run as sub-routine.
Committer's Input: $b \in \{0,1\}$.

Commitmen Phase

1. Receiver $\mathsf{R}_{\mathsf{ExtTok}}$: pick MAC-keys: $k_{\mathsf{rec}}, k_{\mathsf{tok}}$. Create token $\mathcal{T}_{\mathsf{R}}$ implementing the following functionality. On input a tuple $(r||\mathbf{c},\, \sigma_{\mathsf{rec}},\, y)$: if $\mathsf{Vrfy}(k_{\mathsf{rec}}, r||\mathbf{c}, \sigma_{\mathsf{rec}}) = 1$ and $(\mathbf{c} = y$ OR $\mathbf{c} = y \oplus r)$ then output $\sigma_{\mathsf{tok}} = \mathsf{Mac}(k_{\mathsf{tok}}, y)$ else output $\perp$.
(Formally, $\mathsf{R}_{\mathsf{ExtTok}}$ sends $(\mathsf{create}, \mathsf{sid}, \mathsf{R}_{\mathsf{ExtTok}}, \mathsf{C}_{\mathsf{ExtTok}}, \mathcal{T}_{\mathsf{R}})$ to $\mathcal{F}_{\mathsf{wrap}}$). Send $\mathcal{T}_{\mathsf{R}}$ to the sender $\mathsf{C}_{\mathsf{ExtTok}}$.

   Commitment of the Secret Bit: ComBit**.**

2. $\mathsf{C}_{\mathsf{ExtTok}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtTok}}$: run $\langle \mathsf{C}_{\mathsf{IdealTok}}(\mathtt{com}, b), \mathsf{R}_{\mathsf{IdealTok}}(\mathtt{com}) \rangle$ so that $\mathsf{C}_{\mathsf{IdealTok}}$ commits to bit b. Let $(r, \mathbf{c})$ be the transcript of such commitment phase. Let y be the opening of $\mathbf{c}$.

3. Receiver $\mathsf{R}_{\mathsf{ExtTok}}$: compute $\sigma_{\mathsf{rec}} \leftarrow \mathsf{Mac}(k_{\mathsf{rec}}, r||\mathbf{c})$. Send σ_{rec} to Committer $\mathsf{C}_{\mathsf{ExtTok}}$.

4. Committer $\mathsf{C}_{\mathsf{ExtTok}}$: query $\mathcal{T}_{\mathsf{R}}$ with $q = (r||\mathbf{c},\, \sigma_{\mathsf{rec}},\, y)$ (i.e., send $(\mathsf{run}, \mathsf{sid}, \mathsf{C}_{\mathsf{ExtTok}}, \mathcal{T}_{\mathsf{R}}, q)$ to $\mathcal{F}_{\mathsf{wrap}}$) and obtain σ_{tok}. If token $\mathcal{T}_{\mathsf{R}}$ aborts, set $\sigma_{\mathsf{tok}} = 0^n$.

   Commitment of $\mathcal{T}_{\mathsf{R}}$**'s Response:** $\mathsf{ComResp}$**.**
   $\mathsf{C}_{\mathsf{ExtTok}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtTok}}$: run $\langle \mathsf{C}_{\mathsf{IdealTok}}(\mathtt{com}, \sigma_{\mathsf{tok}}), \mathsf{R}_{\mathsf{IdealTok}}(\mathtt{com}) \rangle$ so that $\mathsf{C}_{\mathsf{ExtTok}}$ commits to the response σ_{tok} received from $\mathcal{T}_{\mathsf{R}}$.

Decommitment Phase

1. $\mathsf{C}_{\mathsf{ExtTok}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtTok}}$: opening of both commitments.
   Run $\langle \mathsf{C}_{\mathsf{IdealTok}}(\mathtt{open}, b), \mathsf{R}_{\mathsf{IdealTok}}(\mathtt{open}) \rangle$ and $\langle \mathsf{C}_{\mathsf{pufIdeal}}(\mathtt{open}, \sigma_{\mathsf{rec}}), \mathsf{R}_{\mathsf{pufIdeal}}(\mathtt{open}) \rangle$.

2. Receiver $\mathsf{R}_{\mathsf{ExtTok}}$: If both decommitment are successfully completed, then $\mathsf{R}_{\mathsf{ExtTok}}$ gets the bit b' along with the opening y' for ComBit and string σ'_{tok} for $\mathsf{ComResp}$. If $\mathsf{Vrfy}(k_{\mathsf{tok}}, r||y', \sigma'_{\mathsf{tok}}) = 1$ then $\mathsf{R}_{\mathsf{ExtTok}}$ accept and output b'. Else, reject.

Figure 3.7: ExtTok*: Ideal* **Extractable** *Commitment in the* $\mathcal{F}_{\mathsf{wrap}}$ *model.*

E **runs in polynomial time.** E follows the procedure of the honest receiver, which is efficient.

Simulation. The extractor E follows the procedure of the honest receiver $\mathsf{R}_{\mathsf{ExtTok}}$, and additionally it collects the queries made by $\mathcal{A}$ to $\mathcal{T}_{\mathsf{R}}$. Therefore the view of $\mathcal{A}$ interacting with E is identical to the view of $\mathcal{A}$ interacting with $\mathsf{R}_{\mathsf{ExtTok}}$.

Extraction. We show that, probability that the extractor E outputs $\perp$ (i.e., it fails

Extractor E
E simulates the creation of $\mathcal{T}_{\mathsf{R}}$. Queries made by $\mathcal{A}$ to $\mathcal{T}_{\mathsf{R}}$ are intercepted by E, stored in the variable $\mathcal{Q}$, and then answered faithfully (i.e., by following the code of an honest $\mathcal{T}_{\mathsf{R}}$).
Commitment Phase:
E honestly follows the procedure of $\mathsf{R}_{\mathsf{ExtTok}}$. If the commitment phase is accepting, E proceeds to the extraction phase. Else, it halts. Let $(r, \mathbf{c})$ be the transcript of ComBit.
Extraction Phase:

- If there exists a query q= $(r||\mathbf{c}, \sigma_{\mathsf{rec}}, y) \in \mathcal{Q}$ such that $\mathsf{Vrfy}(k_{\mathsf{rec}}, r||\mathbf{c}, \sigma_{\mathsf{rec}}) = 1$ and $(\mathbf{c} = y\ )$ then output 0.
- If there exists a query q= $(r||\mathbf{c}, \sigma_{\mathsf{rec}}, y) \in \mathcal{Q}$ such that $\mathsf{Vrfy}(k_{\mathsf{rec}}, r||\mathbf{c}, \sigma_{\mathsf{rec}}) = 1$ and $(\mathbf{c} = y \oplus r\ )$ then output 1.
- Case 1) If there exist queries $q_0, q_1 \in \mathcal{Q}$ s.t. $q_0 = (r||\mathbf{c}, \sigma_{\mathsf{rec}}, y_0)$ and $q_1 = (r||\mathbf{c}, \sigma_{\mathsf{rec}}, y_1)$, and $\mathsf{Vrfy}(k_{\mathsf{rec}}, r||\mathbf{c}, \sigma_{\mathsf{rec}}) = 1$ and both y_0, y_1 are valid openings for $(r||\mathbf{c})$ then output $\perp$.
- Case 2) If no queries are accepting, output $\perp$.

Figure 3.8: *E: Extractor associated to* ExtTok.

in extracting the bit) but the adversary $\mathcal{A}$ is instead able to provide an accepting decommitment is negligible. From Fig. 3.8 E fails in the extraction in two cases.

In case 1, the adversary queries the token with two valid openings for the same commitment $\mathbf{c}$. In this case, the commitment $\mathbf{c}$ is not binding. We argue that, due to the binding property of protocol $\mathsf{IdealTok}$, probability that $\mathcal{A}$ later provides an accepting decommitment for $\mathbf{c}$ is negligible. The reason is that, an opening of $\mathbf{c}$ is the pair x, y such that $y = \mathcal{T}_{\mathsf{c}}(x)$. Note also that $|x| = n$ while $|y| = 3n$. The commitment $\mathbf{c}$ is equivocal only if $\mathbf{c} = y_0$ and $r = y_0 \oplus y_1$ for some pair $y_0, y_1 \in \{0,1\}^{3n}$. Since $\mathcal{T}_{\mathsf{c}}$ is sent to the receiver before the string r has been observed, probability that $\mathcal{T}_{\mathsf{c}}$ has been programmed with a pair of strings which exclusive-or is r is negligible. Since x is only n bits, the committer cannot later instruct the token $\mathcal{T}_{\mathsf{c}}$ to answer the value y_b. Thus, probability that $\mathcal{A}$ computes a commitment $\mathbf{c}$ which is equivocal and can be accepted in the decommitment, is negligible as well. Hence, in case 1) extractability is not violated since the extractor fails only when the decommitment will be accepted whp.

Now, consider case 2. Let $r||\mathbf{c}$ be the transcript of the commitment of ComBit. In case 2, the adversary $\mathcal{A}$ did not query the token $\mathcal{T}_{\mathsf{R}}$ with the opening of the commitment $\mathbf{c}$ (but she might have queried with other values). We argue that, in this case, probability that $\mathcal{A}$ provides an accepting decommitment is negligible. Assume, towards a contradiction, that $\mathcal{A}$ provides an accepting decommitment in protocol ExtTok. This means that $\mathcal{A}$ committed to a valid MAC, computed with the key k_{tok}, of the opening y of commitment $\mathbf{c}$, without querying $\mathcal{T}_{\mathsf{R}}$ with y. Now, $\mathcal{A}$ can compute such a MAC in two ways. Either, $\mathcal{A}$ was able to extract the key

k_{tok} by exploiting its access to the *stateless* $\mathcal{T}_{\mathsf{R}}$, or $\mathcal{A}$ was able to forge the MAC under key k_{tok}.

Due to the unconditional one-time unforgeability of MAC and the statistically binding of $\mathsf{IdealTok}$, $\mathcal{A}$ cannot query the token $\mathcal{T}_{\mathsf{R}}$ more then one time (thus extracting the key). Namely, it cannot query $\mathcal{T}_{\mathsf{R}}$ on values which prefix is different from $r||\mathbf{c}, \sigma_{\mathsf{rec}}$ where σ_{rec} is received from the receiver (extractor). This is due to the one-time unforgeability of the MAC used to compute σ_{rec}, and from the fact that $\mathcal{A}$ observes only one MAC computed with k_{rec}. Once the prefix $r||\mathbf{c}$ is fixed, due to the binding of $\mathsf{IdealTok}$, probability that $\mathcal{A}$ can query the token for more then one opening is negligible (except the case in which $\mathbf{c}$ is properly crafted, that we analyzed before). Thus, probability that $\mathcal{A}$ obtains two MACs and extracts the key k_{tok}, is negligible.

Since $\mathcal{A}$ cannot extract k_{tok}, the only case in which it can generate a valid new mac for an opening y, without querying the token, is by forging the MAC. Due to the unforgeability of MAC, this happens with negligible probability.

□

□

3.5 Optimization and reusing of PUFs/Tokens

Recall that protocol $\mathsf{UCComCompiler}$ requires parties to run $2 \cdot (2n)$ extractable commitments. In the security proof of protocol $\mathsf{UCComCompiler}$ we treat an extractable commitment as a black box. Hence, each extractable commitment runs using an independent PUF/Token. However, if the security parameters of PUFs and the functions embedded in the stateless tokens are properly set, then one can re-use the same PUF/Token across all $4n$ executions. Thus, within a *single* execution of $\mathsf{UCComCompiler}$ one can reuse the same PUF/Token for all the extractable commitments. The reason is that, all extractable commitments are run in *parallel*. However, note that the same PUF/Token *cannot* be used among concurrent executions of $\mathsf{UCComCompiler}$. This is consistent with the basic UC-framework, where the same instance of setup assumption cannot be shared among several protocol executions.

Indeed, in the basic UC framework (in contrast with the generalized UC model of [17] for example), a setup assumption cannot be reused among more than one protocol execution. Namely, composability among protocols is guaranteed as long as each protocol uses an independent instance of the setup assumption. For instance, in the standard CRS model, the same CRS cannot be shared by more than one protocol. The reason is that the simulator needs to program the CRS in order to be able to simulate/extract from the protocol.

However, a closer look to the security proof of our compiler, suggests that we do not need the ability of *programming* the setup assumption. Therefore, one can ask whether, our compiler is still secure when the same setup assumption is reused. The answer

is that it depends from the implementation of the underlying extractable commitment scheme. The security of both our extractable commitments relies on the fact that, any response computed by the PUF/Token, sent by the receiver, is revealed to the receiver only in the decommitment phase. This crucially rules out the possibility or reusing the same PUF/Token for concurrent executions. Indeed, consider a malicious receiver sending a token (or PUF) to the committer and starting two protocol executions. Let us denote them session i and session j. Now, consider the case in which in session i the decommitment phase is executed, while session j is still in the commitment phase. Recall that, in the decommitment phase, the committer reveals to the receiver the answer of the receiver's token (or send back the receiver's PUF). It is immediate to see, that if the committer uses the same token is both sessions, the answer of the malicious token can reveal information on the unopened session j. Thus hiding is trivially violated.

Part

Round-Optimal Concurrently Secure Protocols

- Chapter 4 -

Round-Optimal (Black-Box) Constructions for SOA-secure Commitment Schemes

Introduction

Commitment schemes are a fundamental building block in cryptographic protocols. By their usual notion, they satisfy two security properties, namely, hiding and binding. While the binding property guarantees that a committed message can not be opened to two distinct messages, the hiding property ensures that before the decommitment phase begins, no information about the committed message is revealed. Binding and hiding are preserved under concurrent composition, in the sense that even a concurrent malicious sender will not be able to open a committed message in two ways, and even a concurrent malicious receiver will not be able to detect any relevant information about committed messages as long as only commitment phases have been played so far.

In [39], Dwork et al. pointed out a more subtle definition of security for hiding where the malicious receiver is allowed to ask for the opening of only some of the committed messages, with the goal of breaking the hiding property of the remaining committed messages. This notion was captured in [39] via a simulation-based security definition, and is referred to as hiding in presence of selective opening attack (SOA, for short). [39] shows that, in a *trusted setup* setting, it is possible to construct a non-interactive SOA-secure commitment scheme from a trapdoor commitment scheme. Indeed, in the trusted setup the simulator sets the parameters of the trapdoor commitment, thus obviously it knows the trapdoor. However, the fundamental question of whether there exist SOA-secure commitment schemes in the *plain model*, is left open in [39]. We stress that the question is particularly important since commitments are often used in larger protocols, where often only some commitments are opened but the security of the whole scheme still relies on hiding the unopened commitments. For instance, the importance of SOA-secure commitments for constructing zero-knowledge sets is discussed in [46][1].

[1] In [46] some forms of zero-knowledge sets were proposed, and their strongest definition required SOA-secure commitments.

The SOA-security experiment put forth in [39] considers a one-shot commitment phase, in which the receiver gets all commitments in one-shot, picks adaptively a subset of them, and obtains the opening of such subset. Such definition implicitly considers non-interactive commitments and only parallel composition. Subsequent works have explored several extensions/variations of this definition showing possibility and impossibility results. Before proceeding to the discussion of the related work, it is useful to set up the dimensions that will be considered. One dimension is *composition*. As commitment is a two-phase functionality, other than parallel composition, one can consider two kinds of concurrent composition. Concurrent-with-barrier composition (considered in [11, 59]), refers to the setting in which the adversarial receiver can interleave the execution of several commitments, and the execution of decommitments, with the restriction that all commitment phases are played before any decommitment phase begins. Thus, there is a barrier between commitment and decommitment stage. Fully-concurrent composition (considered in [100]) refers to the setting in which the adversary can arbitrarily interleave the execution of the commitment phase of one session with the decommitment of another session (and vice-versa).

Next dimension is the *access to primitive*, namely, if the construction uses a cryptographic primitive as a black-box (in short, BB), or in a non black-box way (in short, NBB).

Another dimension is *simulation*. In this discussion we consider always black-box simulators (if not otherwise specified).

The question of achieving SOA-secure commitments without any set-up was solved affirmatively in [11] by Bellare, Hofheinz, and Yilek, and in Hofheinz [59], who presented an interactive SOA-secure scheme based on non-black-box use of any one-way permutation and with a commitment phase requiring a super-constant number of rounds. The security of such construction is proved in the concurrent-with-barrier setting. [11, 59] also shows that non-interactive SOA-secure commitments which use cryptographic primitives in a black-box way do not exist. The same work introduces the notion of *indistinguishability* under selective opening attacks, that we do not consider in this work. The results of [11, 59] left open several other questions on round optimality and black-box use of the underlying cryptographic primitives.

In TCC 2011 [100], Xiao addressed the above open questions and investigated on how to achieve nearly optimal schemes where optimality concerns both the round complexity and the black-box use of cryptographic primitives. In particular, Xiao addressed SOA-security of commitment schemes for both parallel composition and fully-concurrent composition and provided both possibility and impossibility results, sticking to the simulation-based definition. Concerning positive results, [100] shows a 4-round (resp., $(t + 3)$-round for a t-round statistically-hiding commitment) computationally binding (resp., statistically binding) SOA-secure scheme for parallel composition. Moreover, [100] provides a commitment scheme which is "strong" (the meaning of strong is explained later) SOA-secure in the fully-concurrent setting and requires a logarithmic number of rounds. All such constructions are fully black-box. Concerning impossibility results, [100] shows that 3-round (resp., 4-round) computationally binding (resp., statis-

tically binding) parallel SOA-secure commitment schemes are impossible to achieve. As explained later, in this paper – among other things – we present issues in the proof of security of the constructions shown in [100]. We also show that, the strong security claimed for the construction suggested for the fully-concurrent setting, is actually impossible to achieve, regardless of the round complexity. We contradict the lower bounds claimed in [100] by providing a 3-round fully black-box commitment scheme which is SOA-secure under concurrent-with-barrier composition, which implies parallel composition.

In a subsequent work [102], after our results became publicly available, Xiao showed a black-box construction of 4-round statistically-binding commitment scheme which is SOA-secure under parallel composition. As we shall see later, our 3-round and 4-round schemes are only computationally binding, but are secure in the stronger setting of concurrent-with-barrier composition.

In [101], the same author provides an updated version of [100]. Concerning positive results, [101] includes the $(t+3,1)$-round construction of [100] and shows a new simulation strategy for it. Concerning impossibility results, [101] includes the lower bounds of [100] that are still valid for 2-round (resp., 3 round) computationally hiding and computationally (resp., statistically) binding, parallel SOA-secure commitment scheme with black-box simulators. [101] contains also other contributions of [100] that are not contradicted by this work.

In [8], Bellare et al. proves that existence of CRHFs implies impossibility of non-interactive SOA-secure commitments (regardless of the black-box use of the cryptographic primitives). In fact, they show something even stronger; they show that this impossibility holds even if the simulator is non-black-box and knows the distribution of the message space. An implication of such results is that, standard security does not imply SOA-security. Previous results in [11, 59] only showed the impossibility for the case of black-box reductions.

In [84], Pass and Wee provide several black-box constructions for two-party protocols. Among them it provides constructions for look-ahead trapdoor commitments (in a look-ahead commitment, knowledge of the trapdoor is necessary already in the commitment phase in order for the commitment to be equivocal, in this paper we call such commitments "weak trapdoor commitments"), and trapdoor commitments. Such constructions have not been proved to be SOA-secure commitment schemes, as SOA-security is proven in presence of (at least) parallel composition, while security of the trapdoor commitment of [84] is proved only in the stand-alone setting. In the full version of this paper we discuss how the look-ahead thread of [84] can be plugged in our construction based on weak trapdoor commitments, to obtain a 6-round SOA-secure commitment scheme in the concurrent-with-barrier setting.

Our Contribution

In this work we focus on simulation-based SOA-secure commitment schemes, and we restrict our attention to black-box simulation, and (mainly) black-box access to cryptographic primitives (like in [100]). Firstly, we point out various issues in the claims of [100]. These issues essentially re-open some of the open questions that were claimed

to be answered in [100]. We next show how to solve (in many cases in a nearly optimal way) all of them. Interestingly, our final claims render quite a different state-of-the-art from (and in some cases also in contrast to) the state-of-the-art set by the claims of [100].

In detail, by specifying as (x, y) the round complexity of a commitment scheme when the commitment phase takes x rounds and the decommitment phase takes y rounds, we revisit some claims of [100] and re-open some challenging open questions as follows.

1. The proof in [100] of the non-existence of $(3, 1)$-round schemes assumes implicitly that the sender sends the last message during the commitment phase. We show here that surprisingly this assumption is erroneous, and that one round might be saved in the commitment phase if the receiver goes last. This re-opens the question of the achievability of $(3, 1)$-round SOA-secure schemes, even for just parallel composition.

2. There are issues in the proof of binding and SOA-security of the $(4, 1)$-round scheme of [100] for parallel composition, and it is currently unknown whether the scheme is secure. The same issue in the SOA-security proof exists for the $(t + 3, 1)$-round statistically binding scheme of [100] which is based on any t-round statistically-hiding commitment. Indeed, for both constructions, SOA-security is claimed to follow from the simulation technique of Goldreich-Kahan [48]. The problem is that the simulator of [48] was built for a *stand-alone* zero-knowledge protocol where an atomic sub-protocol is repeated several times in parallel, and the verifier cannot *selectively* abort one of the sub-protocols. Instead in the SOA-setting the adversarial receiver interacts with multiple senders and can decide to abort only a subset of the sessions of its choice adaptively based on the commitment-phase transcript.

   We note that this implies that [100] contains no full proof of a constant round SOA-secure scheme (but we remark that, subsequent to our results, the same author presented a new proof of the $(t + 3, 1)$-round scheme based on statistically-hiding commitment in the work [102, 101]).

3. There is an issue in the proof of security of the fully-concurrent SOA-secure commitment scheme proposed in [100]. The security of such construction is claimed even for the case in which the simulator cannot efficiently sample from the distribution of messages committed to by the honest sender (but needs to query an external party for it).[2] This notion is referred in [100] as "strong" security. This issue in [100] re-opens the possibility of achieving schemes that are strong SOA-secure under fully concurrent composition (for any round complexity).

In this paper we solve the above open problems (still sticking to the notion of black-box simulation as formalized in [100]) as follows.

1. We present a $(3, 1)$-round scheme based on BB use of any trapdoor commitment (TCom, for short), which contradicts the lower bound claimed in [100].

[2] For simplicity, we shall hereafter refer to this case as the simulator not knowing the distribution.

We also provide several constructions based on BB use of various weaker assumptions. We show: a $(3,3)$-round scheme based on BB use of any OWP, a $(4,1)$-round scheme based on BB use of any weak trapdoor commitment (wTCom, for short)[3].

2. We show that when the simulator does not know the distribution of the messages committed to by the honest sender, there exists no scheme that achieves fully concurrent SOA-security, regardless of the round complexity and of the BB use of cryptographic primitives. Thus contradicting the claimed security of the construction given in [100].

3. As a corollary of our $(3,1)$-round scheme based on BB use of any TCom, there exists a $(3,1)$-round scheme based on NBB use of any one-way function (OWF). This improves the round complexity in [11] from logarithmic in the security parameter to only 3 rounds and using minimal complexity-theoretic assumptions. Moreover, we observe that (as a direct consequence from proof techniques in [100]) a $(2,1)$-round SOA-secure scheme is impossible regardless of the use of the underlying cryptographic primitive (for black-box simulation only). Thus, our $(3,1)$-round scheme for black-box simulation is essentially round-optimal.

Notice that both our $(3,1)$-round protocols – the one based on BB use of TCom and the other based on NBB use of OWFs – contradict the impossibility given in [100], that was claimed to hold regardless of the access to the cryptographic primitives.

All the constructions shown in this paper are secure under concurrent-with-barrier composition, which obviously implies parallel composition. Our simulators work for any message distributions, and do not need to know the distribution of the messages committed to by the honest sender. In light of our impossibility for the fully concurrent composition (see Item 2 of the above list), the concurrency achieved by our schemes seems to be optimal for this setting.

As an additional application, we also show that our $(3,1)$-round schemes can be used to obtain non-interactive (concurrent) zero knowledge [40] with 3 rounds of pre-processing. This improves upon [27] where (at least) 3 rounds of interactions are needed both in the pre-processing phase and in the proof phase. Moreover, the simulator of [27] works only with non-aborting verifiers, while our simulator does not have this limitation. This application also establishes usefulness of concurrency-with-barrier setting.

On simulator not knowing message distribution. All our protocols and impossibility results are in the setting where the simulator by itself cannot efficiently sample from the message distribution but needs to query an external oracle for the same. Positive results can only be stronger with this requirement.

[3]This result indeed requires a relaxed definition of trapdoor commitment where the trapdoor is required to be known already during the commitment phase in order to later equivocate. We call it "weak" because any TCom is also a wTCom.

Selective opening, adaptive security, trapdoor commitments and non-malleable commitments. The concept of commitments secure in presence of selective opening attacks is very related to adaptively secure commitments [4], and trapdoor commitments, – in which there exists a trapdoor that allows to open a commitment in many ways. However, they are three different settings.

First, trapdoor commitments are not necessarily SOA-secure (in the plain model). The reason is that trapdoor commitments only guarantee that *there exists* a trapdoor which would allow to equivocate a commitment, however, such trapdoor is clearly not available to a simulator. To achieve SOA-security from a trapdoor commitment scheme one should provide a mechanism for the simulator to get the trapdoor, still not violating binding. The converse moreover is not true, namely, a SOA-secure commitment is not necessarily also a trapdoor commitment. This comes from the fact that the simulator of a SOA-secure commitment could use rewinding capabilities instead of a trapdoor that might not exist at all.

Second, a commitment scheme that is adaptively secure in the parallel composition, is also parallel SOA-secure commitment. The converse is not necessarily true. Namely, a SOA-secure commitment scheme is not necessarily adaptively secure. The reason is that in a selective opening attack, a malicious receiver can "corrupt" the sender in the decommitment phase only, and by definition, it is allowed to see only the openings of the commitments instead of the *whole* state of the sender.

Finally, we stress that our adversary can play as sender only or as receiver only. An interesting question is which type of SOA security can be achieved also against man-in-the-middle attacks. The recent work of [55] gives hope towards a construction of a constant-round non-malleable SOA-secure protocol with black-box simulation and black-box use of any one-way function.

This chapter is divided in three main sections. In the first section we recall the definition of Selective Opening Attacks for Commitment Schemes, and we discuss the "concurrent-with-barrier" variant. In the second section we provide a round-optimal (3 rounds) construction for concurrent SOA-secure Commitment Scheme, that uses only black-box access to any (two-round) Trapdoor Commitment Scheme. We also show how to instantiate such construction with a trapdoor commitment based on any one-way function. In the same section we provide a construction based on black-box uses of weak trapdoor commitments, and it requires 4 rounds (thus is not optimal), assuming the existence of two-round weak trapdoor commitment scheme. All our constructions are not secure in a "fully" concurrent setting, but only in the "concurrent with barrier" setting, that we define.

In the last section of this chapter we prove that this is not a limitation of our constructions, but that achieving (black-box) "fully concurrent" SOA security is impossible.

[4] In such a notion, an adversary can corrupt a party anytime during the protocol execution, obtaining the party's internal state.

4.1 Definitions

We start by providing the definition of Selective Opening Attack for commitment schemes.

Definition 19 (Hiding in presence of Selective Opening Attacks (slight variation of [11, 59]))**.** *Let $k = \mathtt{poly}(n)$, let $\mathcal{B}$ be a k-bit message distribution and $\mathbf{b} \xleftarrow{\$} \mathcal{B}$ be a k-bit vector, let $\mathcal{I} = \{\mathcal{I}_k\}_{k\in\mathbb{N}}$ be a family of sets, where each $\mathcal{I}_k$ is a set of subsets of $[k]$ denoting the set of legal subsets of (indexes of) commitments that the receiver (honest or malicious) is allowed to ask for the opening. A commitment scheme $\mathsf{Com} = (\mathsf{Gen}, \mathsf{C}, \mathsf{R})$ is secure against selective opening attacks if for all k, all sets $I \in \mathcal{I}$, all k-bit message distributions $\mathcal{B}$, all PPT relations $\mathcal{R}$, there exists an expected PPT machine Sim such that for any PPT malicious receiver R^* there exists a negligible function ϵ such that:*

$$\mathbf{Adv}^{\mathsf{soa}}_{\mathsf{Com}} = \left|\Pr\left[\mathbf{Exp}^{\mathrm{real}}_{\mathsf{Com},\mathsf{C},\mathsf{R}^*}(n) \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathrm{ideal}}_{\mathsf{Com},\mathsf{Sim},\mathsf{R}^*}(n) \to 1\right]\right| \leq \epsilon(n)$$

The probability is taken over the choice of the random coins of the parties.

Experiment $\mathbf{Exp}^{\mathrm{real}}_{\mathsf{Com},\mathsf{C},\mathsf{R}^*}(n)$:	**Experiment** $\mathbf{Exp}^{\mathrm{ideal}}_{\mathsf{Com},\mathsf{Sim},\mathsf{R}^*}(n)$:
$pk \xleftarrow{\$} \mathsf{R}^*(1^n)$;	$pk \xleftarrow{\$} \mathsf{R}^*(1^n)$;
$\mathbf{b} \xleftarrow{\$} \mathcal{B}$;	$\mathbf{b} \xleftarrow{\$} \mathcal{B}$;
$I \xleftarrow{\$} \langle \mathsf{C}_i(pk, \mathtt{com}, \mathbf{b}[i])_{i\in[k]}, \mathsf{R}^*(pk, \mathtt{recv})\rangle$;	$I \xleftarrow{\$} \mathsf{Sim}^{\mathsf{R}^*}(pk)$;
$(\cdot, ext) \xleftarrow{\$} \langle \mathsf{C}_i(\mathtt{open})_{i\in I}, \mathsf{R}^*(\mathtt{open})\rangle$;	$ext \xleftarrow{\$} \mathsf{Sim}^{\mathsf{R}^*}(\mathbf{b}[i])_{i\in I}$;
output $\mathcal{R}(I, \mathbf{b}, ext)$.	*output* $\mathcal{R}(I, \mathbf{b}, ext)$.

We denote by $(\cdot, ext) \xleftarrow{\$} \langle \mathsf{C}_i(\cdot), \mathsf{R}^*(\cdot)\rangle$ the output of R^* after having interacted concurrently with k instances of C each one denoted by C_i. In the paper we call an instance as a session. A malicious receiver R^* can run many sessions in concurrency with the following limitation. R^* runs commitment phases concurrently for polynomially many sessions, but it can initiate the first decommitment phase only after the commitment phases of all the sessions have been completed (and therefore after the set of indexes has been requested). This means that the set of indexes I (i.e., the commitments asked to be opened), depends only of the transcript of the commitment phase. We call this definition **concurrent-with-barrier** *(*CwB*, for short), meaning that many commitment phases (decommitment phases) can be run concurrently but the commitment phase of any session cannot be interleaved with the decommitment of any other session. Notice that as in [100], our definition assumes that the honest receiver chooses to open only a subset of the commitments, but this is done independently of the transcript (i.e., $I \xleftarrow{\$} \mathcal{I}$). This means that in the "SOA-commitment" functionality (differently from traditional commitment functionality) the receiver also has an input that corresponds to opening/not opening.*

Remark 2. *In this paper, unless otherwise mentioned, a SOA-secure commitment scheme is a commitment scheme that is SOA-secure under* CwB *composition. In fact, all our positive results are for the setting of* CwB *composition.*

On our choice of definitions

We now discuss the main motivations behind the choices that we made to obtain the above definitions.

Concurrency-with-barrier vs. parallel and concurrent composition. In [100] Xiao provides two main definitions: SOA-security under parallel (PAR) composition and SOA-security under "fully" concurrent composition (CC). In the fully concurrent definition there is no barrier between commitment and decommitment phase: R^* is allowed to interleave the commitment phase of one session with the decommitment phase of another, basically having the power of deciding which decommitment/commitment to execute, depending on the transcript of the commitment *and* decommitment of other sessions. This definition is pretty general, but unfortunately, as we show in this paper, achieving this result is impossible (under the assumption that the simulator does not know the distribution of the messages committed to by the honest sender); this is in contrast to [100] where it is claimed that this definition is achievable. The concurrent-with-barrier composition that we adopted (following [59]) implies security under parallel composition while due to the barrier between commitment and decommitment phase, it is weaker than the fully concurrent definition of [100].

Decommitment phase can be interactive. Following [59] our definition is more general than the one of [100] since it allows also the decommitment phase to be interactive.

Honest party behaviour. We follow [100] in defining the behaviour of the honest receiver i.e, R chooses the subset of commitments to be opened according to some distribution $\mathcal{I}$. To see why this definition makes sense, think about extractable commitments where the sender and receiver engage in many commitments of pairs of shares of a message but finally only one share per pair is required to be opened in the commitment phase.

Concerning the honest sender, we assume that R^* interacts with k independent senders, that are oblivious to each other, and play with input $\mathbf{b}[j]$, while [100] considers a single sender C^k who gets as input the complete k-bit string and plays k independent sessions with R^*. This variation is cosmetic only.

Comparison with the definitions of [11, 59]. In [11, 59] the behaviour of the honest receiver is not explicitly defined, implying that the honest receiver always obtains all the openings. In order to be more general and to make SOA-secure commitments useful in more general scenarios, we deviate from this definition allowing the honest receiver to ask for the opening of a subset of the commitments. Moreover, the set of indexes I chosen by the (possibly malicious) receiver is explicitly given as input to the relation $\mathcal{R}$.

Summing up, the definition that we adopt mainly follows the one of [11, 59] and is more general than the one of [100] in the fact that it allows interaction also during the

decommitment phase, and provides concurrency-with-barrier that implies the definition of security under parallel composition. Moreover, our definition is more general than the one of [59] since it allows also the the honest receiver to choose the commitments to be opened. However, our definition is weaker than the concurrent definition of [100] that however we show to be impossible to achieve (when the distribution of the messages committed by C is unknown to Sim).

4.2 Constructions

In this section we present constructions of a round-optimal SOA-secure commitment scheme based on BB use of trapdoor commitments. In particular we show that if 2-round (where the first round only serves for the receiver to send the public parameters) trapdoor commitment schemes exist[5] then a 3-round commitment scheme that is secure under selective opening attack exists. Under the assumption that *weak* trapdoor commitment schemes exist, in Section 4.2.2 we present a 4-round construction.

The main idea behind both protocols is to require the sender to commit to its private input using a trapdoor commitment scheme and to make the trapdoor information extractable to the black-box simulator. This allows the simulator to cheat in the opening phase without changing the transcript of the commitment phase. Obviously, the parameters of the trapdoor commitment are generated by the receiver (if this was not the case then a malicious sender can cheat in decommitment phase using the trapdoor), and are made extractable through cut-and-choose techniques. In more details, the protocol goes as follows. R runs the generation algorithm of the trapdoor commitment scheme (TCGen) to generate the public parameters used by C to commit to its private bit. To allow extraction of the trapdoor, we require that R provides $2n$ public parameters and C asks to "reveal" the trapdoor of a random n-size subset of them. C will use the remaining n parameters (for which the trapdoors are still hidden) to commit to n shares of its secret input. In this way the equivocation requires the knowledge of one trapdoor only among the set of the remaining n keys that were not revealed. Thus, the simulator first commits to n random bits, then through rewinding threads it will extract from R at least one trapdoor of the remaining unrevealed keys. One trapdoor is sufficient to equivocate one of the shares already committed, and in turn, to decommit to any bit.

In Protocol 1, that uses trapdoor commitments, the simulator can commit without knowing the trapdoor, thus the commitment of the shares can be merged with the cut-and-choose phase, therefore yielding a 3-rounds commitment phase. In the protocol that uses *weak* trapdoor commitments (see Protocol 2 in Section 4.2.2), the simulator needs to extract the trapdoor before committing (since it will be able to equivocate only commitments that are computed using the trapdoor), therefore the commitment must be postponed after the completion of the cut-and-choose phase. This adds one more round to the commitment phase.

[5] [85] is an example of a trapdoor commitment scheme where the public parameters pk are generated by the receiver and sent to the sender in the first round. Given pk, the commitment procedure is non-interactive.

Finally, binding follows straight-forwardly from the binding of the trapdoor commitment scheme used as sub-protocol.

4.2.1 SOA-secure Commitment based on Trapdoor Commitments

Let us denote as $\mathsf{TC} = (\mathsf{TCGen}, \mathsf{C_{TC}}, \mathsf{R_{TC}}, \mathsf{TCFakeDec})$ a trapdoor commitment scheme. In the following we show a protocol $\mathsf{SOACom} = (\mathsf{S_{soa}}, \mathsf{R_{soa}})$ that uses TC as sub-protocol in a black-box fashion. If the commitment phase of TC is non-interactive (given the public parameters in input) then the following construction yields a (3,1)-round commitment scheme. We denote by $\langle \mathsf{C_{TC}}_i^{\bar{ch}_i}, \mathsf{R_{TC}}_i^{\bar{ch}_i} \rangle$ the i-th invocation of sub-protocol TC run with public key $\mathsf{pk}^{\bar{ch}_i}$. Here ch_i denotes the i^{th} challenge for the cut-and-choose, i.e., $\mathsf{S_{soa}}$ computes the trapdoor associated to the key pk^{ch_i}, while it commits to the i^{th} share of the input using key $\mathsf{pk}^{\bar{ch}_i}$ (for which the trapdoor will not be revealed).

Protocol 1. *[(3,1)-Round SOA-Secure Commitment Scheme] [*$\mathsf{SOACom} = (\mathsf{S_{soa}}, \mathsf{R_{soa}})$*]*

Commitment phase.

$\mathsf{R_{soa}}$**:** *For* $i = 1, \ldots, n$*:*

1. $r_i^0, r_i^1 \xleftarrow{\$} \{0,1\}^n$; $(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{TCGen}(r_i^0)$; $(\mathsf{pk}_i^1, \mathsf{sk}_i^1) \leftarrow \mathsf{TCGen}(r_i^1)$;
2. *send* $(\mathsf{pk}_i^0, \mathsf{pk}_i^1)$ *to* $\mathsf{S_{soa}}$;

$\mathsf{S_{soa}}$**:** *On input a bit* b*. Upon receiving* $\{\mathsf{pk}_i^0, \mathsf{pk}_i^1\}_{i\in[n]}$*:*

1. *secret share the bit* b*: for* $i = 1, \ldots, n$*:* $b_i \xleftarrow{\$} \{0,1\}$*, such that* $b = (\bigoplus_{i=1}^n b_i)$;
2. *for* $i = 1, \ldots, n$ *do in parallel:*
   - *send* $ch_i \xleftarrow{\$} \{0,1\}$ *to* $\mathsf{R_{soa}}$;
   - *run* $\langle \mathsf{C_{TC}}_i^{\bar{ch}_i}(\mathsf{pk}_i^{\bar{ch}_i}, \mathtt{com}, b_i), \mathsf{R_{TC}}_i^{\bar{ch}_i}(\mathsf{pk}_i^{\bar{ch}_i}, \mathtt{recv}) \rangle$ *with* $\mathsf{R_{soa}}$;

$\mathsf{R_{soa}}$**:** *Upon receiving* $ch_1, \ldots, ch_n$*: if all commitment phases of protocol* TC *were successfully completed, send* $\{r_i^{ch_i}\}_{i\in[n]}$ *to* $\mathsf{S_{soa}}$;

$\mathsf{S_{soa}}$**:** *Upon receiving* $\{r_i^{ch_i}\}_{i\in[n]}$ *check consistency: for* $i = 1, \ldots, n$*:* $(\mathsf{pk}_i'^{ch_i}, \mathsf{sk}_i'^{ch_i}) \leftarrow \mathsf{TCGen}(r_i^{ch_i})$*; if* $\mathsf{pk}_i'^{ch_i} \neq \mathsf{pk}_i^{ch_i}$ *then abort.*

Decommitment phase.

$\mathsf{S_{soa}}$**:** *for* $i = 1, \ldots, n$*: run* $(\cdot, b_i') \leftarrow \langle \mathsf{C_{TC}}_i^{\bar{ch}_i}(\mathtt{open}), \mathsf{R_{TC}}_i^{\bar{ch}_i}(\mathtt{open}) \rangle$ *with* $\mathsf{R_{soa}}$;

$\mathsf{R_{soa}}$**:** *If all opening phases were successful completed output* $b' \leftarrow \bigoplus_{i=1}^n b_i'$*. Otherwise, output* $\perp$*.*

Theorem 9 (Protocol 1 is secure under selective opening attacks)**.** *If* $\mathsf{TC} = (\mathsf{TCGen}, \mathsf{C_{TC}}, \mathsf{R_{TC}}, \mathsf{TCFakeDec})$ *is a trapdoor commitment scheme, then Protocol 1 is a commitment scheme secure against selective opening attacks.*

Proof. In the following, we prove completeness, binding and hiding under selective-opening-attacks of the $(3,1)$ round protocol presented in Protocol 1.

We refer to the execution of the commitment (resp., decommitment) procedure of the sub-protocol TC as sub-commitment (resp., sub-decommitment). The malicious receiver $\mathsf{R}^*_{\mathsf{soa}}$ plays k concurrent sessions of SOACom; more precisely, she will run k concurrent executions of the commitment phase, and up to $m = |I|$ concurrent decommitment phases.

Completeness. It follows from the completeness of the sub-protocol TC.

Binding. For the binding property it is sufficient to consider the protocol in the stand-alone setting. Therefore we focus on one single session of the protocol SOACom.

We have to show that any PPT malicious sender $\mathsf{C}^*_{\mathsf{soa}}$ is not able to provide two distinct valid openings for the same commitment transcript with non-negligible probability. Note that the decommitment phase consists only of the opening of n sub-commitments for which $\mathsf{C}^*_{\mathsf{soa}}$ has not seen the secret keys [6]. Therefore, if $\mathsf{C}^*_{\mathsf{soa}}$ is able to provide two distinct valid openings, it must be the case that $\mathsf{C}^*_{\mathsf{soa}}$ is able to open at least one of the sub-commitments to both 0 *and* 1, therefore breaking the binding of TC. Due to the binding of TC this event happens with negligible probability.

Formally the reductions goes as follows. Assume that there exists $\mathsf{C}^*_{\mathsf{soa}}$ who breaks the binding of SOACom with non-negligible probability δ. Then there exists at least one pair (i,σ) such that $\mathsf{C}^*_{\mathsf{soa}}$ opens the commitment computed using the public key pk_i^σ in two ways; more formally such that: $(\cdot, b_i) \leftarrow \langle \mathsf{C}_{\mathsf{TC}i}(\mathtt{open}, 0), \mathsf{R}_{\mathsf{TC}i}(\mathtt{open})\rangle$ and $(\cdot, b'_i) \leftarrow \langle \mathsf{C}_{\mathsf{TC}i}(\mathtt{open}, 1), \mathsf{R}_{\mathsf{TC}i}(\mathtt{open})\rangle$ such that $\perp \neq b_i \neq b'_i \neq \perp$. Thus we construct a sender $\mathsf{C}^*_{\mathsf{TC}}$ breaking the binding of the protocol TC with probability $\delta/2n$.

$\mathsf{C}^*_{\mathsf{TC}}$ plays in the experiment $\mathbf{Exp}^{\text{binding}}_{\mathsf{TC}}$ receiving as input the public key pk and interacting with the honest receiver R_{TC}. It runs $\mathsf{C}^*_{\mathsf{soa}}$ as subroutine simulating the receiver $\mathsf{R}_{\mathsf{soa}}$: it randomly picks $i \in [n]$, $\sigma \in \{0,1\}$ and sets $\mathsf{pk}_i^\sigma = pk$, while it honestly generates $2n-1$ pairs of public/secret parameters running TCGen. Finally it sends the $2n-1$ public keys along with pk_i^σ to $\mathsf{C}^*_{\mathsf{soa}}$. Note that this message is distributed identically as the one generated by the honest receiver $\mathsf{R}_{\mathsf{soa}}$. Next, $\mathsf{C}^*_{\mathsf{TC}}$ engages in n sub-commitments with $\mathsf{C}^*_{\mathsf{soa}}$, except that the messages for the sub-commitment in position (i,σ) are forwarded to the honest receiver R_{TC}. When $\mathsf{C}^*_{\mathsf{soa}}$ sends the challenge $ch_1, \ldots, ch_n$: if $ch_i = \sigma$, then $\mathsf{C}^*_{\mathsf{TC}}$ aborts (indeed it is not able to provide the randomness used to generate $pk = pk_i^\sigma$). Otherwise, $\mathsf{C}^*_{\mathsf{TC}}$ answers as the honest receiver $\mathsf{R}_{\mathsf{soa}}$, concluding the commitment phase.

In the opening phase, $\mathsf{C}^*_{\mathsf{TC}}$ is invoked to execute the opening phase with bits 0 and 1, thus it invokes $\mathsf{C}^*_{\mathsf{soa}}$ as well, first with bit 0 and then with bit 1. Each time, $\mathsf{C}^*_{\mathsf{TC}}$ forwards to R_{TC} the i-th sub-decommitment received from $\mathsf{C}^*_{\mathsf{soa}}$. $\mathsf{C}^*_{\mathsf{TC}}$ wins the binding experiment for protocol TC if $\mathsf{C}^*_{\mathsf{soa}}$ provides two distinct openings for the i-th sub-commitment. Therefore $\mathsf{C}^*_{\mathsf{TC}}$ wins the binding game with probability at least $\delta/2n$.

Hiding under selective opening attack. We show a PPT simulator Sim that having

[6] We assume that if $\mathsf{C}^*_{\mathsf{soa}}$ computes a commitment using a public key for which she later asks to see the secret key, she will be caught by the honest receiver.

black-box access to the adversary $\mathsf{R}^*_{\mathsf{soa}}$ generates an output that is distributed as the output generated by the interaction between $\mathsf{R}^*_{\mathsf{soa}}$ and $\mathsf{C}_{\mathsf{soa}}$ in the real game.

Let $m = |I|$ be the number of sessions required by $\mathsf{R}^*_{\mathsf{soa}}$ to be opened. In order to associate the indexes to the sessions opened we use the following notation: we refer to j_l the session corresponding to the l-th index, where $l = 1, \ldots, m$. The simulator works as follows.

SOA-simulator Sim

Initialization phase. Choose random tapes $\mathsf{ran}_{\mathsf{R}}, \mathsf{ran}_{\mathsf{Sim}}$ respectively for $\mathsf{R}^*_{\mathsf{soa}}$ and for the commitment phase. Activate $\mathsf{R}^*_{\mathsf{soa}}(\mathsf{ran}_{\mathsf{R}})$.

Commitment phase (S1). (main thread)

- Upon receiving public keys $\{\mathsf{pk}_i^0, \mathsf{pk}_i^1\}_{i\in[n]}$ from $\mathsf{R}^*_{\mathsf{soa}}$ for some session $j \in [k]$ do:
  1. randomly choose bits $b_1, \ldots, b_n$; $ch_1, \ldots, ch_n$;
  2. for $i = 1, \ldots, n$: commit to b_i with $\mathsf{pk}_i^{\bar{ch}_i}$ by invoking C_{TC} with $\mathsf{R}^*_{\mathsf{soa}}$. Send challenge $ch_1, \ldots, ch_n$ to $\mathsf{R}^*_{\mathsf{soa}}$. If $\mathsf{R}^*_{\mathsf{soa}}$ aborts, then abort session j.
- Upon receiving $\{r_i^{ch_i}\}_{i\in[n]}$ for some session j, check their consistency running $(\mathsf{sk}_i^{ch_i}, \mathsf{pk}_i^{ch_i}) \leftarrow \mathsf{TCGen}(r_i^{ch_i})$. If the check fails, abort session j. Otherwise store the secret keys $\{\mathsf{sk}_i^{ch_i}\}_{i\in[n]}$ for session j.

Commitment phase completion. When the commitment phase of all k sessions is completed, Sim obtains the set of indexes I from $\mathsf{R}^*_{\mathsf{soa}}$. Sim then outputs I and obtains $\{\mathbf{b}[j]\}_{j\in I}$ from the experiment.

Extraction phase (rewinding thread).
For $l = 1, \ldots, m$; for session $j_l \in I$ that was not-aborted in the main thread do:

1. activate $\mathsf{R}^*_{\mathsf{soa}}$ with randomness $\mathsf{ran}_{\mathsf{R}}$ and use $\mathsf{ran}_{\mathsf{Sim}}$ to execute all the sessions except j_l.
2. in session j_l, uniformly choose bits $ch'_1, \ldots, ch'_n$ and compute the sub-commitments as in Step **S1** (note that the view of $\mathsf{R}^*_{\mathsf{soa}}$ generated in this step is distributed identically as in the main thread). If $\mathsf{R}^*_{\mathsf{soa}}$ aborts then go to Step 1. If $\mathsf{R}^*_{\mathsf{soa}}$ starts new sessions, follow instructions as in Step **S1**.
3. When $\mathsf{R}^*_{\mathsf{soa}}$ replies with $\{r_i^{ch'_i}\}_{i\in[n]}$: if strings $ch'_1 \ldots ch'_n$, $ch_1 \ldots ch_n$ (the challenge used for session j_l in the main thread) are equal then abort the simulation. Else, if there exists an $r_i^{ch'_i}$ generating a valid pair $(\mathsf{sk}_i^{ch'_i}, \mathsf{pk}_i^{ch'_i})$ where $\mathsf{pk}_i^{ch'_i}$ appeared in the j_l-th session of main thread, and $ch'_i \neq ch_i$ then return $\mathsf{sk}_i^{ch'_i}$. Otherwise go to Step 1.

Extraction completion. If Sim reaches this point, then for every(not-aborted) session $j_l \in I$ there is at least one i for which Sim obtained both trapdoors sk_i^0 and sk_i^1.

Decommitment phase (S2) (main thread). Run $\mathsf{R}^*_{\mathsf{soa}}(\mathsf{ran}_{\mathsf{R}})$ till the completion of the commitment phase using $\mathsf{ran}_{\mathsf{Sim}}$. Then for non-aborted sessions $j_l \in I$, with $l \in [m]$, let $\mathbf{b}[j_l]$ be the bit to decommit to in the session j_l, let i be the index such that Sim obtained $\mathsf{sk}_i^0, \mathsf{sk}_i^1$ for the session j_l.

When $\mathsf{R}^*_{\mathsf{soa}}$ asks for the decommitment of the j_l-th session proceed as follows:

1. for all $l \neq i$ honestly run the sub-decommitment algorithm, i.e., $\langle \mathsf{C}_{\mathsf{TC}_l}^{\bar{ch}_l}(\mathtt{open}), \mathsf{R}_{\mathsf{TC}_l}^{\bar{ch}_l}(\mathtt{open})\rangle$, where ch_l is the challenge sent in the commitment phase of the main thread. Compute $b'_i \leftarrow (\bigoplus_{l=1}^{n-1} b_l) \oplus \mathbf{b}[j]$.
2. to open the i-th sub-commitment run the sub-fake-decommitment algorithm using the trapdoor information $\mathsf{sk}_i^{\bar{ch}_i}$, i.e., $(\cdot, b'_i) \leftarrow \langle \mathsf{TCFakeDec}(\mathsf{sk}_i^{\bar{ch}}, \mathtt{open}, b'_i), \mathsf{R}_{\mathsf{TC}_i}^{\bar{ch}_i}(\mathtt{open})\rangle$; If $\mathsf{R}^*_{\mathsf{soa}}$ aborts, then aborts this session.

Finally, output whatever $\mathsf{R}^*_{\mathsf{soa}}$ outputs.

For simplicity we are assuming that the underlying trapdoor commitment TC satisfies the trapdoorness property for any $b^\star \in \{0,1\}$ (Pedersen commitment [85] achieves this property). In case the trapdoorness holds only for a specific bit $b^\star$, then the above simulator should be tweaked only in the extraction phase adding a further condition. That is, when extracting a new secret key $\mathsf{sk}_i^{ch_i}$ for a session j, the simulator considers the extraction phase successfully completed for such session, only if the commitment in position (i, ch_i) is a commitment of $b^\star$. If this is not the case, then Sim continues the rewinding threads. It is easy to see that this further condition is satisfied w.h.p and similar analysis that we show for the simpler simulator apply.

Proposition 1. *The simulator* Sim *runs in expected polynomial time in* n.

Proof. Sim consists of three phases: commitment phase, extraction phase, decommitment phase. Let us denote as $\mathtt{t}_c$, $\mathtt{t}_d$, $\mathtt{t}_{fd}$, $\mathtt{t}_g$ the running times required to execute, respectively, the sub-commitment, the sub-decommitment, the fake-decommitment and the generator algorithm of the protocol TC. By definition of TC all these running times are polynomial in n.

In the commitment phase, for each session, Sim executes $2n$ sub-commitments and verifies the validity of the response of $\mathsf{R}^*_{\mathsf{soa}}$ running the generation algorithm TCGen n times. Plus there is a linear time due to the choice of the random challenge. Thus, the running time of the commitment phase for one session is: $\mathtt{t}_{\mathtt{SimCom}} = \mathtt{t}_c \cdot 2n + \mathtt{t}_g \cdot n + \Theta(n)$. Hence, the total running time for the commitment phase is $k \cdot \mathtt{t}_{\mathtt{SimCom}}$, that is polynomial.

After the completion of the commitment phase, Sim launches the rewinding threads, so that it extracts at least one trapdoor for each session j_l that has been asked for the decommitment. The number of decommitments asked by $\mathsf{R}^*_{\mathsf{soa}}$ is $m = |I|$.

Sim extracts the trapdoors one session at time, hence it runs the extraction procedure at most m times. For each (non-aborting) session j_l, Sim forces upon $\mathsf{R}^*_{\mathsf{soa}}$ the same transcript generated in the main thread, and it changes only the random challenge and

the public keys used in the commitment phase of session l. Thus, the view of $\mathsf{R}^*_{\mathsf{soa}}$ is distributed identically as in the main thread. Then it repeats this procedure rewinding $\mathsf{R}^*_{\mathsf{soa}}$ until either a secret has been extracted or the fresh challenge chosen in a rewinding thread is identical to the challenge sent in the main thread. The probability of the latter event in any of the sessions is at most $\mathtt{poly}(n)/2^n$ and thus is negligible.

More formally, let us denote by $\zeta^{j_l} = \zeta(\mathsf{ran}_{\mathsf{R}}, j_l)$ the probability that $\mathsf{R}^*_{\mathsf{soa}}$, activated with randomness $\mathsf{ran}_{\mathsf{R}}$, correctly responded to the challenge (i.e., $ch_1, \ldots, ch_n$) sent by Sim in the commitment phase of the j_l-th session in the main thread. In each rewinding thread the probability to get another correct answer (for some $ch'_i, \ldots, ch'_n$) is still ζ^{j_l}.

Thus, for each session j_l, the expected number of rewinds needed for the extraction of the trapdoor is bounded by $1/\zeta^{j_l}$. Moreover, upon receiving a new challenge, $\mathsf{R}^*_{\mathsf{soa}}$ may initiate new sessions (i.e., sessions that did not appear in the main thread). In this case Sim follows the same procedure of the commitment phase, therefore each possibly new session initiated during the rewind takes time at most $\mathsf{t}_{\mathtt{SimCom}}$. Upon each rewind $\mathsf{R}^*_{\mathsf{soa}}$ may initiate at most a polynomial number of sessions, therefore the additional work of the simulator for each rewind, that we denote by $\mathsf{t}_{\mathtt{rew}}$, is bounded by $\mathtt{poly}(n) \cdot \mathsf{t}_{\mathtt{SimCom}}$. Obviously, once the simulator extracts the trapdoor for the target session, the new sessions are discarded.

In the decommitment phase Sim executes $n-1$ sub-decommitments plus one execution of the fake-decommitment algorithm, for at most m sessions. Each decommitment phase takes running time: $\mathsf{t}_{\mathtt{SimDec}} = (n-1) \cdot \mathsf{t}_d + \mathsf{t}_{fd}$, that is polynomial in n.

Hence, the total expected running time is:

$$\mathsf{t}_{\mathsf{Sim}} = k \cdot \mathsf{t}_{\mathtt{SimCom}} + \sum_{l=1}^{m} \zeta^{j_l} \Big[\frac{1}{\zeta^{j_l}} \cdot \mathsf{t}_{\mathtt{rew}} + \mathsf{t}_{\mathtt{SimDec}} \Big] = \mathtt{poly}(n).$$

□

Proposition 2. *The distribution of the output of the simulator* Sim *having black-box access to* $\mathsf{R}^*_{\mathsf{soa}}$ *is computationally indistinguishable from the output of* $\mathsf{R}^*_{\mathsf{soa}}$ *interacting with the real sender* $\mathsf{C}_{\mathsf{soa}}$.

Proof. Consider the following hybrids:

H_0**:** In this experiment Sim has as input the bit-vector $\mathbf{b} \leftarrow \mathcal{B}$ and follows the code of the honest sender $\mathsf{S}_{\mathsf{soa}}$. This is the real game.

H_1**:** This hybrid is the same as H_0 except that here, after Sim receives the set I from $\mathsf{R}^*_{\mathsf{soa}}$ (upon the completion of the commitment phase), it launches the extraction phase. That is, for each non-aborted session $j_l \in I$, Sim launches the extraction phase to obtain the trapdoor $\mathsf{sk}_i^{\bar{ch}_i}$ for at least one $i \in [n]$. Possible new sessions initiated by $\mathsf{R}^*_{\mathsf{soa}}$ in the rewinding attempts of the extraction phase are handled by running the honest sender procedure using the knowledge of $\mathbf{b}$. The extracted trapdoors are never used by Sim. We now argue that H_0 and H_1 are indistinguishable. First note that, the extraction phase is initiated only for non-aborting sessions, therefore, only

for those in which $\mathsf{R}^*_{\mathsf{soa}}$ correctly completed the commitment phase with non-zero probability. Then note that the view of $\mathsf{R}^*_{\mathsf{soa}}$ in the rewinding thread is distributed identically to her view in the commitment phase of the main thread. Thus the only differences between H_0 and H_1 are: 1) in H_1 Sim runs in expected polynomial time and 2) in H_1 Sim aborts with higher probability due to the possible aborts in the rewinding threads.

Concerning 1), the expected running time is not a problem since we are only interested in the output of the experiment, and it will not be an issue in the reductions shown for the next hybrids since rewinding threads that take longer than a fixed polynomial can be truncated without perturbing the non-negligible probability of success. Concerning 2), observe that in the rewinding threads an abort happens when Sim picks a random challenge that is equal to the challenge sent in the main thread, and it happens with at most negligible probability $\mathtt{poly}(n)/2^n$. Therefore, hybrids H_0 and H_1 are statistically indistinguishable.

In the following experiments we first deal with the (potential) new sessions initiated by $\mathsf{R}^*_{\mathsf{soa}}$ in the rewinding threads. Recall that Sim tries to extract the secret for one session at time. For each rewinding attempt for the extraction of a session j_l, $\mathsf{R}^*_{\mathsf{soa}}$ may initiate several new sessions. We indicate with max_{j_l} the maximum number of new sessions started during the rewinding threads for the session j_l. Note that, for new sessions started during rewinding threads, Sim is never required to provide the decommitment (therefore those sessions do not need to be rewound).

$H_2^{j_l,s+1}$**: for** $l = 1, \ldots, m$**; for** $s = 0, \ldots, \mathsf{max}_{j_l} - 1$**.** In hybrid $H_2^{j_l,s+1}$ Sim works as in experiment $H_2^{j_l,s}$ except that, in the $(s+1)$th *new* session started by $\mathsf{R}^*_{\mathsf{soa}}$ during the extraction phase launched for session j_l, Sim commits to a random bit.

Toward showing the indistinguishability of $H_2^{j_l,s}$ and $H_2^{j_l,s+1}$, we first show that $H_2^{j_l,s}$ is indistinguishable from a hybrid $\bar{H}_2^{j_l,s+1}$ where the bit committed to in the $(s+1)$-th new session is the negation of the bit used in $H_2^{j_l,s}$.

More precisely, hybrid $\bar{H}_2^{j_l,s+1}$ is the same as hybrid $H_2^{j_l,s}$ except that in the commitment phase of the $(s+1)$-th new session initiated by $\mathsf{R}^*_{\mathsf{soa}}$ in the extraction phase of session j_l, one of the sub-commitments hides the opposite bit such that the sum of the shares of all sub-commitments gives $1 - \mathbf{b}[t]$. We denote the index of the $(s+1)$-th new session by t, where $1 \leq t \leq k$. More specifically, let $b_1, \ldots, b_n$ be the shares of bit $\mathbf{b}[t]$ in the experiment $\bar{H}_2^{j_l,s+1}$, Sim flips one of the shares, i.e. there exists one i such that Sim, differently from the experiment $H_2^{j_l,s}$ commits to $\bar{b}_i$, thus in turn committing to $\bar{\mathbf{b}}[t]$.

Assume that there exists a distinguisher D_{soa} that is able to tell apart hybrid $\bar{H}_2^{j_l,s+1}$ from $H_2^{j_l,s}$ then it is possible to construct a distinguisher who breaks the hiding of the commitment scheme TC. The reduction works as follows. $\mathsf{R}^*_{\mathsf{TC}}$ runs $\mathsf{R}^*_{\mathsf{soa}}$ as a subroutine and simulates Sim as in experiment $H_2^{j_l,s}$ except that, in the new session t, upon receiving the public keys from $\mathsf{R}^*_{\mathsf{soa}}$ it forwards $\mathsf{pk}_i^{\bar{ch}_i}$ to the

external sender $\mathsf{C_{TC}}$. We stress that $\mathsf{pk}_i^{\bar{ch}_i}$ could be maliciously chosen. This is not a problem since the hiding experiment is defined for any public parameter pk^* maliciously chosen by R^*.

Upon receiving the sub-commitment from $\mathsf{C_{TC}}$ for the public key $\mathsf{pk}_i^{\bar{ch}_i}$, $\mathsf{R}^*_{\mathsf{TC}}$ randomly chooses $n-1$ random shares $b_1, \ldots, b_n - 1$ and honestly executes $n-1$ sub-commitments using the remaining public parameters received from $\mathsf{R}^*_{\mathsf{soa}}$. Then it forwards all the sub-commitments to $\mathsf{R}^*_{\mathsf{soa}}$. Finally $\mathsf{R}^*_{\mathsf{TC}}$ forwards the output of the experiment to D_{soa} and outputs whatever D_{soa} outputs xored with $\bigoplus_{l\in[n],l\neq i} b_l$.

Now, let b_i such that $\bigoplus_{l\in[n]} b_l = \mathbf{b}[t]$, if $\mathsf{C_{TC}}$ has committed to the share b_i, then the view generated by $\mathsf{R}^*_{\mathsf{TC}}$ is distributed identically to hybrid $H_2^{j_l,s}$. Otherwise, if $\mathsf{C_{TC}}$ has committed to bit $1-b_i$ then the view generated is distributed identically to hybrid $\bar{H}_2^{j_l,s+1}$. By the hiding of protocol TC, we have that $H_2^{j_l,s}$ and $\bar{H}_2^{j_l,s+1}$ are indistinguishable.

Now, in $H_2^{j_l,s+1}$ the bit committed in session t (i.e., the $(s+1)$-th new session) is a random bit, and therefore the output of any distinguisher on $H_2^{j_l,s+1}$ will be indistinguishable from the one of $H_2^{j_l,s}$ and $\bar{H}_2^{j_l,s+1}$.

Therefore, $H_1 = H_2^{1,0}$ and $H_2^{j_m,\mathsf{max}_{j_m}}$ are indistinguishable.

$H_3^{j_{l+1}}$**: for** $l = 0, \ldots, m-1$**:** In this sequence of hybrids Sim performs the decommitment phase of the sessions that have been asked for by $\mathsf{R}^*_{\mathsf{soa}}$, using the trapdoor extracted in the extraction phase, therefore, technically Sim can open to any bit.

More precisely, hybrid $H_3^{j_{l+1}}$ is the same as hybrid $H_3^{j_l}$ except that in $H_3^{j_{l+1}}$ in the decommitment phase of the j_{l+1}-th session Sim uses the trapdoor $\mathsf{sk}_i^{\bar{ch}_i}$ (for some $i \in [n]$) extracted for this session. That is, in session j_{l+1} Sim honestly performs $n-1$ sub-decommitments while the i-th sub-decommitment is executed running $\mathsf{TCFakeDec}$ on input the bit b_i that is computed as follows: $b_i \leftarrow \bigoplus_{l\in[n],l\neq i} b_l^{\bar{ch}_l} \oplus \mathbf{b}[j_{l+1}]$. Note that now in the opening, b_i depends of the actual input of the sender. However, in this experiment the share b_i computed in the commitment phase is identical to the share b_i given in input to the algorithm $\mathsf{TCFakeDec}$. More precisely, $\mathsf{TCFakeDec}$ is not used to open to a different bit, but to the very same bit.

Assume that there exists a distinguisher D_{soa} able to tell apart $H_3^{j_{l+1}}$ from $H_3^{j_l}$ with non-negligible probability δ, then it is possible to construct an adversary $\mathsf{R}^*_{\mathsf{TC}}$ against the trapdoor property of TC. $\mathsf{R}^*_{\mathsf{TC}}$ runs in the experiment $\mathbf{Exp}^{\mathsf{trap/Com}}_{\mathsf{TC}}$ against a sender $\mathsf{C_{TC}}$ and works as follows.

It runs $\mathsf{R}^*_{\mathsf{soa}}$ as subroutine, it randomly chooses a session s, and then proceeds as follows: for the commitment phase it simulates all sessions as in experiment $H_3^{j_l}$ except the session s. In such session, $\mathsf{R}^*_{\mathsf{TC}}$ after having received the public keys from $\mathsf{R}^*_{\mathsf{soa}}$, performs the secret sharing of the bit $\mathbf{b}[j_{l+1}] = b_1 \oplus \cdots \oplus b_n$, picks challenge $ch_1, \ldots, ch_n$, an index $i \in [n]$, and forwards $\mathsf{pk}_i^{\bar{ch}_i}$,$b_i$ to $\mathsf{C_{TC}}$. Then it engages in $n-1$ sub-commitments of TC with $\mathsf{R}^*_{\mathsf{soa}}$ for the commitment of b_l, with $l \neq i$ while

for the i-th sub-commitment it forwards the messages received by $\mathsf{C_{TC}}$. Once the commitment phase is over, $\mathsf{R^*_{TC}}$ runs the extraction phase. If it does not get the secret key $\mathsf{sk}_i^{\bar{ch}_i}$ it aborts. Otherwise, it continues executing the decommitment phase.

In the decommitment phase $\mathsf{R^*_{soa}}$ asks the opening of m sessions: $\{j_1, \ldots, j_m\}$; if $s \neq j_{l+1}$ then $\mathsf{R^*_{TC}}$ aborts. Otherwise it computes the decommitment phase of the first l sessions (i.e., $j_1, \ldots, j_l$) as in hybrid $H_3^{j_l}$, while for the decommitment of session s (that is the j_{l+1}-th session asked for opening) $\mathsf{R^*_{TC}}$ sends $\mathsf{sk}_i^{\bar{ch}_i}$ to $\mathsf{C_{TC}}$ and forwards the decommitment received by $\mathsf{C_{TC}}$ to $\mathsf{R^*_{soa}}$ along with the remaining $n-1$ sub-decommitments honestly computed. Finally $\mathsf{R^*_{TC}}$ forwards the output of $\mathsf{R^*_{soa}}$ to D_{soa} and outputs what D_{soa} outputs. Now, if in the j_{l+1}-th session, the i-th sub-decommitment was computed by algorithm $\mathsf{TCFakeDec}$, then the view of $\mathsf{R^*_{soa}}$ is distributed identically as hybrid $H_3^{j_{l+1}}$, otherwise, if it was computed using the honest sender procedure, then the view is distributed according to hybrid $H_3^{j_l}$. Therefore, if D_{soa} distinguishes the two experiments with non-negligible advantage δ then $\mathsf{R^*_{TC}}$ wins the game $\mathbf{Exp}_{\mathsf{TC}}^{\mathsf{trap/Com}}$ with advantage at least $\delta \cdot \frac{m}{k} \cdot \frac{1}{2n}$ that is still non-negligible therefore breaking the trapdoor property of TC. Hence, $H_3^{j_{l+1}}$ and $H_3^{j_l}$ are computationally indistinguishable.

Therefore $H_2^{j_m,\mathsf{max}_{j_m}} = H_3^{j_0}$ and $H_3^{j_m}$ are computationally indistinguishable.

H_4^{j+1}**: for** $j = 0, \ldots, k-1$**.** In this sequence of hybrids Sim performs the commitment phase committing to random bits instead of using the vector $\mathbf{b}$.

More precisely, hybrid H_4^{j+1} is the same as H_4^j except that in H_4^{j+1} Sim performs the commitment phase of the session j committing to a random bit instead of $\mathbf{b}[j]$. Due to hiding of the commitment scheme TC, and following the same arguments for distinguishability of hybrids $H_2^{j_l,s}$ and $H_2^{j_l,s+1}$, hybrids H_4^{j+1} and H_4^j are indistinguishable.

By noticing $H_4^0 = H_3^{j_m}$ and that H_4^k corresponds to the game played by the simulator, we have that the claim holds.

□

SOA-secure Scheme based on NBB use of OWFs. We observe that, by instantiating Protocol 1 with the Feige-Shamir trapdoor commitment scheme described in Section 1.1.1, one can obtain a (3,1) SOA-secure scheme with non-black-box access to OWFs.

4.2.2 SOA-secure Commitment based on Weak Trapdoor Commitments

Let us denote as $\mathsf{wTCom} = (\mathsf{wTCGen}, \mathsf{C_{TC}}, \mathsf{R_{TC}}, \mathsf{TCFakeCom}, \mathsf{TCFakeDec})$ a weak-trapdoor commitment scheme. In the following we show a construction $\mathsf{SOACom} =$

$(\mathsf{S}_{\mathsf{soa}}, \mathsf{R}_{\mathsf{soa}})$ that uses wTCom as a black-box. If wTCom is (2,1)-round weak trapdoor commitment scheme the following construction is a (4,1)-round commitment scheme. As in the previous construction, we indicate with $\langle \mathsf{C}_{\mathsf{TC}}^{\sigma}, \mathsf{R}_{\mathsf{TC}}^{\sigma} \rangle$ the i-th invocation of the algorithms of protocol wTCom using the public key pk_i^{σ}. The sender $\mathsf{S}_{\mathsf{soa}}$ has as input a bit b.

Protocol 2. *[(4,1)-round SOA-secure commitment scheme based on BB use of weak trapdoor commitment] [*$\mathsf{SOACom} = (\mathsf{S}_{\mathsf{soa}}, \mathsf{R}_{\mathsf{soa}})$*]*

Commitment phase.

$\mathsf{R}_{\mathsf{soa}}$**:** *For* $i = 1, \ldots, n$*:*

1. r_i^0, $r_i^1 \xleftarrow{\$} \{0,1\}^n$; $(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{wTCGen}(r_i^0)$;$(\mathsf{pk}_i^1, \mathsf{sk}_i^1) \leftarrow \mathsf{wTCGen}(r_i^1)$;
2. *send* $\{\mathsf{pk}_i^0, \mathsf{pk}_i^1\}$ *to* $\mathsf{S}_{\mathsf{soa}}$;

$\mathsf{S}_{\mathsf{soa}}$**:** *Upon receiving* $\{\mathsf{pk}_i^0, \mathsf{pk}_i^1\}_{i\in[n]}$*: send* $ch_1, \ldots, ch_n$ *to* $\mathsf{R}_{\mathsf{soa}}$ *where* $ch_i \xleftarrow{\$} \{0,1\}$;

$\mathsf{R}_{\mathsf{soa}}$**:** *Upon receiving* $ch_1, \ldots, ch_n$ *send* $\{r_i^{ch_i}\}_{i\in[n]}$ *to* $\mathsf{S}_{\mathsf{soa}}$;

$\mathsf{S}_{\mathsf{soa}}$**:** *Upon receiving* $\{r_i^{ch_i}\}_{i\in[n]}$*:*

1. **check consistency***: for* $i = 1, \ldots, n$*:* $(\mathsf{pk}_i'^{ch_i}, \mathsf{sk}_i'^{ch_i}) \leftarrow \mathsf{wTCGen}(r_i^{ch_i})$*; if* $\mathsf{pk}_i'^{ch_i} \neq \mathsf{pk}_i^{ch_i}$ *then* `ABORT`.
2. *secret share the bit* b*: for* $i = 1, \ldots, n$*:* $b_i \xleftarrow{\$} \{0,1\}$*, such that* $b = (\bigoplus_{i=1}^{n} b_i)$;
3. *run* $\langle \mathsf{C}_{\mathsf{TC}_i}^{\bar{ch}_i}(\mathsf{pk}_i^{\bar{ch}_i}, \mathtt{com}, b_i), \mathsf{R}_{\mathsf{TC}_i}^{\bar{ch}_i}(\mathsf{pk}_i^{\bar{ch}_i}, \mathtt{recv}) \rangle$ *with* $\mathsf{R}_{\mathsf{soa}}$;

Decommitment phase.

$\mathsf{S}_{\mathsf{soa}}$**:** *For* $i = 1, \ldots, n$*: run* $(\cdot, b_i') \leftarrow \langle \mathsf{C}_{\mathsf{TC}_i}^{\bar{ch}_i}(\mathtt{open}), \mathsf{R}_{\mathsf{TC}_i}^{\bar{ch}_i}(\mathtt{open}) \rangle$ *with* $\mathsf{R}_{\mathsf{soa}}$;

$\mathsf{R}_{\mathsf{soa}}$**:** *If all opening phases were successfully completed output* $b' \leftarrow \bigoplus_{i=1}^{n} b_i'$*. Else output* $\perp$.

The above protocol can be instantiated with the *weak* trapdoor commitment scheme based on BB access to OWPs shown in [84]. In such a protocol the commitment is interactive, and follows the commit-challenge-response structure. The commitment is such that if the sender knows the challenge in advance, it can commit in a way that allows equivocation. In such a scheme the trapdoor is the challenge sent by the receiver, and in turn, the public parameter is the (statistically hiding) commitment of the challenge. One can plug such protocol in our (4,1)-round SOA-secure protocol and obtain a (6,1)-round protocol (the commitment phase in [84] is interactive).

Theorem 10 (Protocol 2 is secure under selective opening attacks)**.** *If* $\mathsf{wTCom} = (\mathsf{wTCGen}, \mathsf{C}_{\mathsf{TC}}, \mathsf{R}_{\mathsf{TC}}, \mathsf{TCFakeCom}, \mathsf{TCFakeDec})$ *is a weak-trapdoor commitment scheme, then protocol 2 is a commitment scheme secure under selective opening attacks.*

Completeness. It follows from the completeness of the sub-protocol wTCom.

Binding. The binding proof follows the same logic of the one provided for Protocol 1, and is therefore omitted.

Hiding under selective opening attack. We show a PPT simulator Sim that having black-box access to the adversary $\mathsf{R}^*_{\mathsf{soa}}$ generates an output that is distributed as the output generated from the interaction between $\mathsf{R}^*_{\mathsf{soa}}$ and the real world sender $\mathsf{C}_{\mathsf{soa}}$. The simulator works as follows.

SOA-simulator Sim

Initialization phase. Choose random tape $\mathsf{ran}_{\mathsf{R}}$ and activate $\mathsf{R}^*_{\mathsf{soa}}(\mathsf{ran}_{\mathsf{R}})$.

Commitment phase. Main thread.

1. Upon receiving public keys $\{\mathsf{pk}_i^0, \mathsf{pk}_i^1\}_{i\in[n]}$ for some session $j \in [k]$: pick a random n-bit string $ch_1, \ldots, ch_n$ and send it to $\mathsf{R}^*_{\mathsf{soa}}$. Label this point as **1-j**.
2. Upon receiving $\{r_i^{ch_i}\}_{i\in[n]}$ for some session j, check their consistency by running $(\mathsf{sk}_i^{ch_i}, \mathsf{pk}_i^{ch_i}) \leftarrow \mathsf{TCGen}(r_i^{ch_i})$. If the check fails, abort session j. In case there exists a secret key $(\mathsf{sk}_i^{\bar{ch}_i})$ (for some $i \in [n]$) already stored for session j, then the trapdoor for session j has been extracted, thus go to step 4. Else go to step 3.
3. Extraction of secret keys for session j: start **rewinding threads** to extract $\mathsf{sk}_i^{\bar{ch}_i}$ for some $i \in [n]$:
   (a) rewind $\mathsf{R}^*_{\mathsf{soa}}$ up to point **1-j**.
   (b) send a randomly chosen challenge string $ch'_1, \ldots, ch'_n$ to $\mathsf{R}^*_{\mathsf{soa}}$. If $\mathsf{R}^*_{\mathsf{soa}}$ aborts, go to Step 3a.
   (c) if $\mathsf{R}^*_{\mathsf{soa}}$ starts new commitment sessions, follow the commitment procedure of the honest sender $\mathsf{S}_{\mathsf{soa}}$ committing to a random bit.
   (d) when $\mathsf{R}^*_{\mathsf{soa}}$ replies with secrets $\{r_i^{ch'_i}\}_{i\in[n]}$ for the session j: if the random strings $ch'_1, \ldots, ch'_n$ and $ch_1, \ldots, ch_n$ are equal, abort. Else, if there exists at least one $r_i^{ch'_i}$ generating a valid pair $(\mathsf{sk}_i^{ch'_i}, \mathsf{pk}_i^{ch'_i})$ where $\mathsf{pk}_i^{ch'_i}$ was received in Step **1-j** and $ch'_i \neq ch_i$ store the secret key $(\mathsf{sk}_i^{ch'_i})$ for session j, rewind $\mathsf{R}^*_{\mathsf{soa}}$ up to Step 2 and return. Otherwise go to Step 3a.
4. Commitment of session j: On input the pair $(\mathsf{sk}_i^0, \mathsf{sk}_i^1)$ proceeds with the commitments for the session j;
   (a) randomly choose bits $b_1, \ldots, b_n$;
   (b) for all bits b_l s.t. $l \neq i$ honestly run the sub-commitment algorithm: $(\cdot, b_l) \overset{\$}{\leftarrow} \langle \mathsf{C}_{\mathsf{TC}_l}^{\bar{ch}_l}(\mathtt{open}), \mathsf{R}_{\mathsf{TC}_l}^{\bar{ch}_l}(\mathtt{open})\rangle$ with $\mathsf{R}^*_{\mathsf{soa}}$ where ch_l is the challenge sent in Step **1-j**.
   (c) for the i-th sub-commitment, run the fake commitment procedure using the secret key $\mathsf{sk}_i^{\bar{ch}_i}$: $\langle \mathsf{TCFakeCom}\,(\mathsf{pk}_i^{\bar{ch}_i}, \mathsf{sk}_i^{\bar{ch}_i}, \mathtt{com}), \mathsf{R}_{\mathsf{TC}_i}^{\bar{ch}_i}(\mathsf{pk}_i^{\bar{ch}_i}, \mathtt{recv})\rangle$. If $\mathsf{R}^*_{\mathsf{soa}}$ aborts, then aborts this session.

Commitment phase completion. When the commitment phase is completed, Sim obtains the set of indexes I from $\mathsf{R}^*_{\mathsf{soa}}$ and obtains $\{\mathbf{b}[j]\}_{j\in I}$ from the experiment.

Decommitment phase. When $\mathsf{R}^*_{\mathsf{soa}}$ asks for the opening of session $j \in I$, run n sub-decommitments as follows:

1. for all sub-commitments $l \neq i$ honestly run the sub-decommitment algorithm: $(\cdot, b_l) \xleftarrow{\$} \langle \mathsf{C}_{\mathsf{TC}_l}^{\bar{ch}_l}(\mathsf{open}), \mathsf{R}_{\mathsf{TC}_l}^{\bar{ch}_l}(\mathsf{open})\rangle$, where ch_l is the challenge sent in the commitment phase. Compute $b'_i \leftarrow (\bigoplus_{l\in[n-1]} b_l) \oplus \mathbf{b}[j]$.
2. for the i-th sub-commitment run the fake-sub-decommitment algorithm using the trapdoor information $\mathsf{sk}_i^{\bar{ch}_i}$: $(\cdot, b'_i) \leftarrow \langle \mathsf{TCFakeDec}(\mathsf{sk}_i^{\bar{ch}}, \mathsf{open}, b'_i), \mathsf{R}_{\mathsf{TC}_i}^{\bar{ch}_i}(\mathsf{open})\rangle$. If $\mathsf{R}^*_{\mathsf{soa}}$ aborts, then abort this session.

Finally, output whatever $\mathsf{R}^*_{\mathsf{soa}}$ outputs.

Proposition 3. *The simulator* Sim *runs in expected polynomial time in* n.

Proof. As the simulator strategy mainly follows the strategy of the simulator before the analysis of the running time follows the same arguments shown in Proposition 1. □

Proposition 4. *The distribution of the output of simulator* Sim *having black-box access to* $\mathsf{R}^*_{\mathsf{soa}}$ *is computationally close to the output of* $\mathsf{R}^*_{\mathsf{soa}}$ *interacting with real sender* $\mathsf{S}_{\mathsf{soa}}$.

Proof. Consider the following hybrids:

H_0**:** In this experiment Sim has in input the bit-vector $\mathbf{b} \leftarrow \mathcal{B}$ and follows the code of the honest sender $\mathsf{S}_{\mathsf{soa}}$. This is the real game.

In the following hybrids, we denote by κ the number of sessions opened by $\mathsf{R}^*_{\mathsf{soa}}$ in the main-thread only. We denote by j_l with $l = 0, \ldots, \kappa - 1$ the l-th session opened in the main thread for which Sim obtains a valid second message (i.e., the values $\{r_i^{ch_i}\}_{i\in[n]}$) from $\mathsf{R}^*_{\mathsf{soa}}$, and thus it has to extract the secret key in order to be able to compute the sub-commitments in such session.

$H_1^{j_{l+1}}$ **(for** $l = 0, \ldots, \kappa - 1$**):** Experiment $H_1^{j_{l+1}}$ is the same as experiment $H_1^{j_l}$ except that in the $(l+1)$-th session for which Sim obtains the values $\{r_i^{ch_i}\}_{i\in[n]}$ from $\mathsf{R}^*_{\mathsf{soa}}$, it launches the extraction phase to extract the trapdoor $\mathsf{sk}_i^{\bar{ch}_i}$ for some $i \in [n]$ for the session j_{l+1}. In the new sessions initiated by $\mathsf{R}^*_{\mathsf{soa}}$ during the rewinds, Sim runs as the honest sender using the knowledge of $\mathbf{b}$. However, the extracted trapdoor is never used by Sim. We now argue that $H_1^{j_l}$ and $H_1^{j_{l+1}}$ are indistinguishable. First note that, the extraction phase is initiated only if $\mathsf{R}^*_{\mathsf{soa}}$ correctly completed the second step of the protocol with non-zero probability. Then note that the view of $\mathsf{R}^*_{\mathsf{soa}}$ in the rewinding thread is distributed identically to her view in the commitment phase. Thus the only differences between the two experiments is that 1) in $H_1^{j_l+1}$ Sim runs in expected polynomial time, this is not an issue since we are only interested in the output of the experiment (and it will not be a problem in the

reductions shown for the next hybrids since rewinding threads that take more time than a fixed polynomial, can be truncated, without perturbing the non-negligible probability of success); 2) in $H_1^{j_l+1}$ Sim aborts with higher probability due to the possible aborts in the rewinding threads. These aborts happen when Sim picks a random challenge that is equal to the challenge sent in the commitment phase. This event happens with negligible probability ($\mathtt{poly}(n)/2^n$). Thus $H_1^{j_l}$ and $H_1^{j_l+1}$ are statistically indistinguishable. Note that $H_1^{j_0} = H_0$ and $H_1^{j_\kappa} = H_2^{j_0}$

$H_2^{j_l,s+1}$**: for** $l = 1, \ldots, \kappa$**; for** $s = 0, \ldots, \mathsf{max}_{j_l} - 1$**.** In hybrid $H_2^{j_l,s+1}$ Sim works as in experiment $H_2^{j_l,s}$ except that, in the $(s+1)$th *new* session started by $\mathsf{R}^*_{\mathsf{soa}}$ in the rewinding threads (recall that in each rewinding thread the view of $\mathsf{R}^*_{\mathsf{soa}}$ changes) for session j_l, Sim commits to a random bit. Toward showing the indistinguishability of $H_2^{j_l,s}$ and $H_2^{j_l,s+1}$, we first show that $H_2^{j_l,s}$ is indistinguishable from an hybrid $\bar{H}_2^{j_l,s+1}$ where the bit committed in the $(s+1)$-th new session is the opposite bit used in $H_2^{j_l,s}$.

More precisely hybrid $\bar{H}_2^{j_l,s+1}$ is the same as hybrid $H_2^{j_l,s}$ except that in the commitment phase of the $(s+1)$-th new session, which index (in the range between 1 and k of all sessions played in the current view) we denote by t, initiated by $\mathsf{R}^*_{\mathsf{soa}}$ in the extraction phase of session j_l, the last sub-commitment hides the opposite bit such that the sum of the shares of all sub-commitments gives $1 - \mathbf{b}[t]$. More specifically, let $b_1, \ldots, b_n$ the shares of bit $\mathbf{b}[t]$, in experiment $\bar{H}_2^{j_l,s+1}$, Sim flips one of the shares, i.e. there exist one i such that Sim, differently from the experiment $H_2^{j_l,s}$ commits to $\bar{b}_i$, thus in turn committing to $\bar{\mathbf{b}}[t]$.

Assume that there exists a distinguisher D_{soa} that is able to tell apart hybrid $\bar{H}_2^{j_l,s+1}$ from $H_2^{j_l,s}$ then it is possible to construct a distinguisher who breaks the hiding of the commitment scheme wTCom. The reduction works as follows. $\mathsf{R}^*_{\mathsf{wTCom}}$ simulates Sim as in experiment $H_2^{j_l,s}$ except that in the new session t, it proceeds as follows: after having received the public keys from $\mathsf{R}^*_{\mathsf{soa}}$, it picks a random n-bit string $ch_i, \ldots, ch_n$ and an index i and it forwards $\mathsf{pk}_i^{ch_i}$ to the external sender $\mathsf{C}_{\mathsf{wTCom}}$.

Upon receiving the sub-commitment from $\mathsf{C}_{\mathsf{wTCom}}$ for the public key $\mathsf{pk}_i^{\bar{ch}_i}$, $\mathsf{R}^*_{\mathsf{TC}}$ randomly chooses $n-1$ random bits $b_1, \ldots, b_n - 1$ and honestly executes $n-1$ sub-commitments using the remaining public parameters received from $\mathsf{R}^*_{\mathsf{soa}}$. Then it forwards all the sub-commitments to $\mathsf{R}^*_{\mathsf{soa}}$. Finally $\mathsf{R}^*_{\mathsf{TC}}$ forwards the output of the experiment to D_{soa} and outputs whatever D_{soa} outputs xored with $\bigoplus_{l\in[n],l\neq i} b_l$.

Now, let b_i such that $\bigoplus_{l\in[n]} b_l = \mathbf{b}[t]$, if $\mathsf{C}_{\mathsf{wTCom}}$ has committed to the share b_i, then the view generated by $\mathsf{R}^*_{\mathsf{TC}}$ is distributed identically to hybrid $H_2^{j_l,s}$. Otherwise, if $\mathsf{C}_{\mathsf{wTCom}}$ has committed to bit $1-b_i$ then the view generated is distributed identically to hybrid $\bar{H}_2^{j_l,s+1}$. By the hiding of protocol wTCom, it holds hat $H_2^{j_l,s}$ and $\bar{H}_2^{j_l,s+1}$ are indistinguishable.

Now, in $H_2^{j_l,s+1}$ the bit committed in session t (i.e., the $(s+1)$-th new session)

is a random bit, and therefore the output of any distinguisher on $H_2^{j_l,s+1}$ will be indistinguishable from the one of $H_2^{j_l,s}$ and $\bar{H}_2^{j_l,s+1}$.

Therefore, $H_1 = H_2^{j_1,0}$ and $H_2^{j_m,\mathsf{max}_{j_m}}$ are indistinguishable.

$H_3^{j_{l+1}}$**: for** $l = 0, \ldots, \kappa - 1$**:** In this sequence of hybrids Sim uses the trapdoor extracted in the extraction phase. In each session, it performs the commitment/decommitment phase by using the algorithms $\mathsf{TCFakeCom}/\mathsf{TCFakeDec}$ for one of the n the sub-commitments. Therefore, in this hybrid Sim does not use the knowledge of $\mathbf{b}$ anymore.

More precisely, hybrid $H_3^{j_{l+1}}$ is the same as hybrid $H_3^{j_l}$ except that in $H_3^{j_{l+1}}$ in the decommitment phase of the j_{l+1}-th session Sim uses the trapdoor $\mathsf{sk}_i^{\bar{ch}_i}$ (for some $i \in [n]$) extracted for this session. That is, in session j_{l+1} Sim honestly performs $n-1$ sub-decommitments while the i-th sub-commitment is computed invoking the fake-sub-commitment algorithm $\mathsf{TCFakeCom}$ and the sub-decommitment (if session j_{l+1} will be asked to be opened) is computed invoking the trapdoor algorithm $\mathsf{TCFakeDec}$ on input the bit b_i computed as follows: $b'_i \leftarrow \bigoplus_{l \in [n], l \neq i} b_l^{\bar{ch}_l} \oplus \mathbf{b}[j_{l+1}]$. Note that now in the opening, b_i depends of the actual input of the sender.

Assume there exists a distinguisher D_{soa} who is able to tell apart experiment $H_3^{j_{l+1}}$ from $H_3^{j_l}$ then it is possible to construct a distinguisher $\mathsf{R}^*_{\mathsf{wTCom}}$ for the weak trapdoor property of wTCom. $\mathsf{R}^*_{\mathsf{wTCom}}$ is running in the experiment $\mathbf{Exp}^{\mathsf{wTrap/Com}}_{\mathsf{wTCom}}$ trying to distinguish whether the messages received from sender $\mathsf{C}_{\mathsf{wTCom}}$ are computed using the honest or the fake algorithm. $\mathsf{R}^*_{\mathsf{wTCom}}$ works as follows: it runs $\mathsf{R}^*_{\mathsf{soa}}$ as subroutine simulating Sim as in experiment $H_3^{j_l}$ except that in session j_{l+1}, after having obtained (from the extraction phase) the trapdoor $\mathsf{sk}^{\bar{ch}_i}$ for some $i \in [n]$ proceeds as follows. It performs the secret sharing of the bit $\mathbf{b}[j_{l+1}] = b_1, \ldots, b_n$ and forwards $\mathsf{pk}_i^{\bar{ch}_i}$,$\mathsf{sk}_i^{\bar{ch}_i}$, b_i to $\mathsf{C}_{\mathsf{wTCom}}$ (note that if the sender $\mathsf{C}_{\mathsf{wTCom}}$ is running $\mathsf{TCFakeCom}$ the bit b_i is ignored and is given only in the decommitment phase to the algorithm $\mathsf{TCFakeDec}$.). Then $\mathsf{R}^*_{\mathsf{wTCom}}$ honestly computes the sub-commitments for bits b_l, for $l \neq i$, while for the i-th sub-commitment it forwards the commitment received from $\mathsf{C}_{\mathsf{wTCom}}$. When the commitment phase is completed $\mathsf{R}^*_{\mathsf{wTCom}}$ obtains the set I from $\mathsf{R}^*_{\mathsf{soa}}$, if the set does not contain the session j_{l+1}, it aborts. Otherwise, $\mathsf{R}^*_{\mathsf{wTCom}}$ performs the decommitment phase of the session j_{l+1} as follows: it honestly opens the sub-commitments in position $i \neq l$, while it forwards the decommitment received from $\mathsf{C}_{\mathsf{wTCom}}$ in position i. Finally $\mathsf{R}^*_{\mathsf{wTCom}}$ forwards the output of $\mathsf{R}^*_{\mathsf{soa}}$ to D_{soa} and outputs whatever D_{soa} outputs. Now, if the i-th sub-commitment/sub-decommitment was computed by using the trapdoor $\mathsf{sk}_i^{\bar{ch}_i}$, then the view of $\mathsf{R}^*_{\mathsf{soa}}$ is distributed identically as hybrid $H_3^{j_{l+1}}$, otherwise, if the sub-commitment/sub-decommitment was honestly computed then the view is distributed according to hybrid $H_3^{j_l}$. Therefore, if D_{soa} distinguishes the two experiments with non-negligible advantage δ then $\mathsf{R}^*_{\mathsf{wTCom}}$ wins the game $\mathbf{Exp}^{\mathsf{trap/Com}}_{\mathsf{wTCom}}$ with advantage $\frac{\delta m}{k}$ that is still non-negligible, therefore breaking the trapdoor

property of wTCom.

Hence, $H_3^{j_{l+1}}$ and $H_3^{j_l}$ are computationally indistinguishable.

Therefore $H_2^{j_m,\mathsf{max}_{j_m}} = H_3^{j_0}$ and $H_3^{j_k}$ are computationally indistinguishable.

By noticing that $H_3^{j_k}$ corresponds to the game played by the simulator, we have that the claim holds.

□

This concludes the proof of the Theorem 10. □

4.3 Impossibility of Fully Concurrent Black-Box SOA-Secure Commitments

The protocols presented so far achieve security under concurrent-with-barrier composition in the "strong" sense, that is, assuming that the simulator does not know the distribution of the messages committed to by the sender. The last question to answer is whether there exist protocols that actually achieve the definition of security under strong fully (i.e., without barrier) concurrent composition (as defined in [100]), or if the concurrent-with-barrier security definition is the best one can hope to achieve (when black-box simulation is taken into account). In this section we show that in contrast to the claim of Theorem 3.5 of [100], the strong fully concurrent security definition of [100] is impossible to achieve. This holds regardless of the round complexity of the protocol [7] and of the black-box use of cryptographic primitives. Under the assumption that OWFs exist, the only requirements that we use for the impossibility is that the simulator is black-box and does not know the distribution of the messages committed by the sender. Both requirements are already specified in the strong fully concurrent security definition of [100]. In the following, we first recall the definition provided in [100] for completeness, then we give the intuitions behind the proof.

4.3.1 Definition

Let $\mathcal{B}$, $\mathcal{I}$, k be as defined in Definition 19 and $\mathbf{b} \xleftarrow{\$} \mathcal{B}$ be the input given to the honest sender C.

Security is defined as comparison of two experiments. In the real world experiment R^* interacts with C in k concurrent sessions and is allowed to pick the set I incrementally. For example, the receiver can generate one commit-phase transcript, ask the sender to decommit that instance, then use this information in its interaction to generate the second commit-phase transcript, and so forth. The output of this experiment is defined as $\langle \mathsf{C}^k(\mathbf{b}), \mathsf{R}^* \rangle = (\tau^k, I, \{b_i, w_i\}_{i \in I})$, where τ^k is the transcript of the commitment phases of the k concurrent sessions, I is the final subset of positions asked incrementally by R^* during the execution, $\{b_i, w_i\}_{i \in I}$ are pairs such that b_i is the bit committed to and

[7] This is therefore different from the case of concurrent zero knowledge [20, 87].

w_i is the opening data (recall that this definition assumes that the decommitment is non-interactive, however our impossibility result holds even for protocol with interactive decommitment phase). In the ideal game, an expected PPT simulator Sim without the knowledge of the vector $\mathbf{b}$ interacts with R^* while incrementally giving as output a set I for which it receives the bits $\{b_i\}_{i\in I}$. Finally, Sim outputs τ^k and $\{b_i, w_i\}_{i\in I}$. This can be seen as if Sim has access to an oracle $\mathcal{O}$ that knows the vector $\mathbf{b}$ and answers to a query j with the value $\mathbf{b}[j]$. The output $(\mathsf{Sim}_k^{\mathsf{R}^*}|\mathbf{b})$ of this experiment is $(\tau^k, I, \{b_i, w_i\}_{i\in I})$ where τ^k,$\{b_i, w_i\}_{i\in I}$ are outputs of Sim while I is the set containing the indexes queried by Sim to the oracle $\mathcal{O}$.

A bit commitment scheme Π is SOA-secure under concurrent composition if, for every $\mathcal{I}, \mathcal{B}$ and k, there exists Sim such that for all R^* it holds that $\langle \mathsf{C}^k(\mathbf{b}), \mathsf{R}^*\rangle$ and $(\mathsf{Sim}_k^{\mathsf{R}^*}|\mathbf{b})$ are computationally indistinguishable. As stated in [100], the above definition is the weakest one since the order of the quantifier is such that the simulator *knows* the message distribution $\mathcal{B}$. Such a definition is motivated by the fact the it makes the lower bounds proved in [100] stronger. If instead there exists Sim that works for all $\mathcal{B}, \mathcal{I}$ and R^* then the protocol is said SOA-secure under fully concurrent composition. All the constructions shown in [100] are claimed to achieve this strong(er) definition in which the message distribution $\mathcal{B}$ is not known by Sim. The same definition can be extended to concurrent SOA-secure string commitment scheme.

From the definition shown above note the following. The set I given as output in the ideal game is not controlled by Sim but corresponds to the set of queries made by Sim to the oracle. If this was not the case then a simulator can just ask for all the openings at the very beginning, perfectly simulate the sender and give as output the set asked by R^* instead of the queries actually made to the oracle. This restriction essentially means that Sim should be very careful in querying the oracle since each query will appear in the final output and there is no possibility to abort or rewind the simulation, as instead it is possible with the transcript of the conversation with R^*.

Theorem 11. *If OWF exists, then no string commitment scheme can be SOA-secure under fully concurrent composition.*

4.3.2 Impossibility Proof

Our proof consists in adapting a proof provided by Lindell in [68].[68] shows that there exist functionalities for which proving that a protocol is secure under m-concurrent composition using a black-box simulator requires that the protocol has at least m rounds. As corollary it holds that for such functionalities unbounded concurrency proved using a black-box simulator is impossible to achieve. Such a theorem cannot be directly applied to the case of SOA-secure commitments since it is provided only for two functionalities in which both parties have private inputs, such as, blind signatures and OT functionalities. In the setting of SOA-secure commitments the receiver has no private input and there is no ideal functionality involved. In our proof, we convert the role of the oracle $\mathcal{O}$ into the role of the functionality, and when deriving the contradiction we do not break the privacy of the receiver but the correctness of the protocol (i.e. the binding).

The proof is based on the following two observations. First of all, since the simulator is black-box the only advantage that it can exploit to carry out a successful simulation is to rewind the adversary. Moreover, rewinds must be effective, in the sense that upon each rewind the simulator should change the transcript in order to "extract" information from the adversary (obviously if the transcript is not changed then the rewind is useless). The second crucial observation is that in SOA the adversary R^* chooses the sessions to decommitment adaptively on the transcript, and in order to obtain the string to provide the decommitment, Sim must query an external oracle (recall that we are considering the strong definition in which the simulator does not know the message distribution). Thus, changing the transcript in the rewinding yields to different sessions asked by R^*, and in turns more queries made by Sim to the oracle. Such additional queries are caused only by the rewinding attempts and they do not appear in the real world execution. However, the distribution of I due to Sim in the ideal game should not be distinguishable from the one due to R^* interacting with the sender in the real world. Thus the idea of the proof is to show that there exists an adversarial strategy that makes the rewinding attempts of any black-box Sim ineffective, unless Sim queries the oracle a number of time that is distinguishable from the number of openings asked by the adversary in the real experiment. Then the next step is to show that if nevertheless there exists a simulator that is able to deal with such an adversary (without rewinding), then such a simulator can be used by a malicious sender to break the binding of the protocol.

We are now ready to provide a formal argument.

Proof. In our proof we assume the existence of OWFs, which is implied by the existence of any commitment scheme. Let $\Pi = (\mathsf{C}, \mathsf{R})$ be a r-round string-commitment protocol that is SOA-secure (with a black-box simulation strategy) under concurrent composition. By definition there exists a black-box simulator Sim that for all R^* is able to produce a view $(\tau^k, I, \{b_i, w_i\}_{i \in I})$ that is indistinguishable from the view of the interaction between R^* and C. In the next part of the proof we will use Sim as a sub-routine to contradict in strict polynomial time some hardness assumptions. Since Sim is only expected polynomial time, we are implicitly assuming that we run it up to some strict polynomial number of steps (obviously greater than the expected polynomial time), and our results will still work since Sim is often successful in that polynomial number of steps.

The formal proof consists of the following steps. First, we define the family of message distributions $\mathcal{B}$. Then we define a pair of adversaries $\mathsf{R}_0^*, \mathsf{R}_1^*$ and we show that such adversaries make the rewinding strategy of any black-box Sim ineffective for one protocol execution. Still, by the concurrent SOA-security of Π it must be the case that Sim is able to successfully carry out the simulation of such execution even without rewinding R_p^*, for $p = 0, 1$. Finally we construct a malicious sender that runs such a simulator as sub-routine to break the binding of Π, thus contradicting the hypothesis that Π is a commitment scheme.

Distribution $\mathcal{B}$. Recall that $\mathcal{B}$ is the set of distribution over $(\{0,1\}^n)^k$, where k is the number of sessions. In order to define our particular set $\mathcal{B}$ we use a signature scheme.

A signature scheme is a tuple of algorithms $(\mathsf{GenSign}, \mathsf{Sign}, \mathsf{Vrfy})$ where $\mathsf{GenSign}$ is the generation algorithm that on input the security parameter outputs a pair vk, sk where sk is the secret key and vk is the verification key. Sign is a randomized signing algorithm and Vrfy is the verification algorithm. The correctness requirement states that for all $(vk, sk) \xleftarrow{\$} \mathsf{GenSign}(1^n)$, all messages x, all randomness r, $\mathsf{Vrfy}(vk, x, \mathsf{Sign}(sk, x, r)) = 1$. The security requirement (called unforgeability) states that no efficient algorithm M, even after having seen polynomially many signatures of messages of her choice is able to produce a new pair (x^*, σ^*) such that $\mathsf{Vrfy}(vk, x^*, \sigma^*) = 1$ without knowing sk. We define our set $\mathcal{B} = \{\mathcal{B}_{sk}\}_{sk \xleftarrow{\$} \mathsf{GenSign}(1^n)}$ where:

$$\mathcal{B}_{sk} = \{\sigma_1, \ldots, \sigma_k : \sigma_i = \mathsf{Sign}(sk, i, r), \text{for } r \in \{0,1\}^n\}$$

Thus we define the message space as a set of signatures under a secret key sk, in particular the message committed to in session j is the signature of j under sk. Then we assume that the adversarial receiver is given in auxiliary input the verification key vk, such that, it is allowed to check whether messages obtained in the opening phase belong to the distribution $\mathcal{B}_{sk}$. Notice that, having defined $\mathcal{B}$ in this way, we have that a query made by Sim to oracle $\mathcal{O}$ to receive the opening of an index j, correspond to a query to a signing oracle $\mathcal{O}^{sk}$ for the signature $\mathsf{Sign}(sk, j, r)$. Such definition of $\mathcal{B}$ allows us to formally claim that in order to simulate the honest sender Sim is forced to ask $\mathcal{O}$, unless it is able to forge the signatures. Having fixed $\mathcal{B}_{sk}$ we are ready to define the adversaries $\{\mathsf{R}^*_p\}_{p \in \{0,1\}}$.

Adversary's strategy. R^*_p, for $p = 0, 1$ runs $k = r(2n) + 1$ protocol executions, where r is the number of rounds of protocol Π. Let us denote by $\Pi_1, \ldots, \Pi_k$ the k protocol executions, by $\Pi_j(i)$ the i-th round of the protocol Π_j, and by $\Pi_j(i)_{\mathsf{C}}$ and $\Pi_j(i)_{\mathsf{R}}$ the messages sent by C and R in that round of the protocol. Let $F_k : \{0,1\}^* \to \{0,1\}^n$ be a PRF for $k \xleftarrow{\$} \{0,1\}^n$.

The adversary's strategy is the following. R^*_p plays the first round of Π_1 (i.e $\Pi_1(1)$) following the procedure of the honest receiver and then it starts a block of $2n$ executions in parallel $(\Pi_2, \ldots, \Pi_{2n+1})$. In this block of executions R^*_p honestly completes the $2n$ commitment phases while the selection for the sessions to open (denoted as I_1) is computed as follows: consider the $2n$ executions as a sequence of n pairs (each pair consists of left and right execution) then: 1. R^*_p computes an n-bit string $s_1 \leftarrow F_k(\Pi_1(1)_{\mathsf{C}})$; 2. consider the ℓ-th pair with $\ell \in [n]$, among the n pairs of executions, R^*_p asks to open the left execution of the ℓ-th pair if the ℓ-th bit of s_1 is 0 and the right execution otherwise. We denote this process of selecting the set of positions I according to the output of F_k by $I_1 \rightleftharpoons F_k(\Pi_1(1)_{\mathsf{C}})$. Then R^*_p sends I_1 to C and obtains the openings $\sigma_{j_1}, \ldots, \sigma_{j_n}$ with $j_i \in I_1$ and checks if $\mathsf{Vrfy}(vk, j_i, \sigma_{j_i}) = 1$ for all $j_i \in I_1$. If any check fails, then it aborts. Otherwise, R^* runs $\Pi_1(2)$ (the second round of Π_1) and starts another block of $2n$ executions till completion as described before. In general, after the i^{th} round of Π_1, R^*_p initiates a block of $2n$ parallel executions of Π $(\Pi_{2(i-1)n+i+1}, \ldots, \Pi_{2in+i})$ and selects the subset of positions I_i according to $F(\Pi_1(i)_{\mathsf{C}})$.

Finally, when the commitment phase of the execution Π_1 is completed, R^*_p asks for the opening with probability p. In Fig. 4.1 we show the diagram of the schedule.

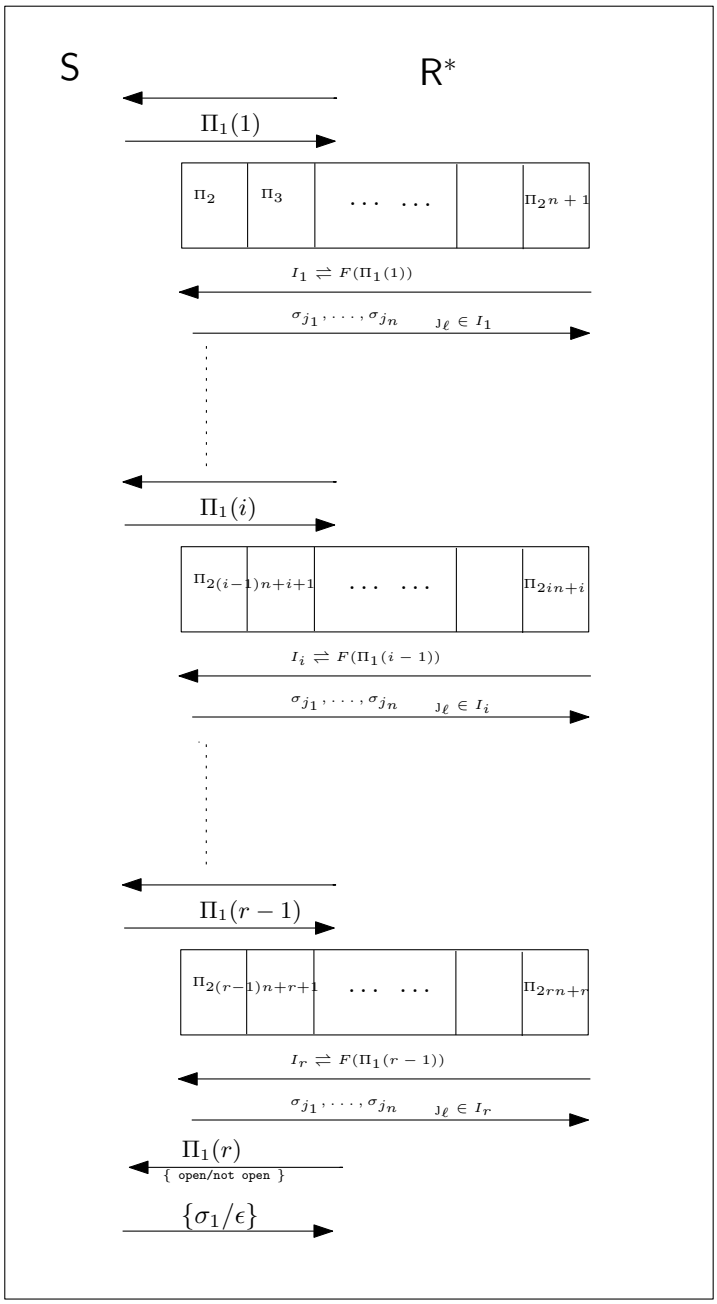

Figure 4.1: *Adversarial strategy of* $\mathsf{R}_0^*/\mathsf{R}_1^*$.

Ideal-World Simulator. By the assumption that Π is black-box secure we have that there exists a black-box simulator Sim such that the output of the ideal execution with Sim is indistinguishable from the result of a real execution of $\{\mathsf{R}_p^*\}_{p\in\{0,1\}}$ running $\Pi_1, \ldots, \Pi_k$ with C. As black-box simulator Sim must work for all malicious R^* and

therefore also for R_0^* and R_1^* defined above. Recall that Sim is given oracle access to R_p^*, namely Sim is given a next message function that receives a sequence of messages and computes R_p^*'s response. If any prefix of the query is such that R_p^* would abort upon that message, then the output for the entire query is $\perp$. We now prove a special property of *all* oracle calls in which R_p^* does *not* abort.

Claim 4.3.1. *For every i let Q_i be the set of all queries sent by Sim to R_p^* during the ideal world execution which includes all messages from the block of executions activated after the message $\Pi_1(i)_{\mathsf{C}}$ (i.e., from $\Pi_{2(i-1)n+i+1}$ to $\Pi_{2(i)n+i}$) and where R_p^* does not abort[8]. Then, the same message $\Pi_1(i)_{\mathsf{C}}$ appears in every $q \in Q_i$, except with negligible probability.*

Proof. The proof of this claim follows almost identically the claim proved in [68] except that here we use signature scheme instead of one-time signature scheme and between each round of the protocol Π_1 here the adversary opens a bunch of $2n$ new executions instead of only one. As discussed in the paragraph of the proof intuition, the main reason we have these differences is that in the SOA-secure setting we cannot exploit the fact that both parties have private inputs. The proof is based on the unforgeability of the signature scheme and the collision resistance property of the PRF.

First, we claim that Sim does not produce two oracle queries q and q' containing $\Pi_1(i)_{\mathsf{C}} \neq \Pi_1(i)'_{\mathsf{C}}$ such that $F_k(\Pi_1(i)_{\mathsf{C}}) = F_k(\Pi_1(i)'_{\mathsf{C}})$ with non negligible probability. Otherwise we can construct a machine M that has oracle access to a random function and distinguishes if the oracle is F_k (for a random k) or a truly random function. Machine M works by emulating the entire experiment between R^* and Sim except that instead of R^* computing $s_i = F_k(\Pi_1(i)_{\mathsf{C}})$, machine M queries the oracle with $\Pi_1(i)_{\mathsf{C}}$. Now, if the oracle is F_k then the emulation is perfect. Therefore with non-negligible probability M obtains from Sim two messages $\Pi_1(i)_{\mathsf{C}} \neq \Pi_1(i)'_{\mathsf{C}}$ such that the oracle responses is the same. On the other hand, if the oracle is a truly random function then the collision happens with negligible probability. Thus M distinguishes with non-negligible probability.

Now we prove that Sim cannot produce two non-aborting queries q, q' such that q contains message $\Pi_1(i)_{\mathsf{C}}$ and q' contains message $\Pi_1(i)'_{\mathsf{C}}$. This claim follows from the unforgeability of the signature scheme, and by the fact that Sim cannot ask to the oracle more openings that R_p^* would ask in the real world attack. The intuition is the following. By the proof given above, if there exist q, q' such that $\Pi_1(i)_{\mathsf{C}} \neq \Pi_1(i)'_{\mathsf{C}}$ then it must be that $s_i \leftarrow F_k(\Pi_1(i)_{\mathsf{C}}) \neq s'_i \leftarrow F_k(\Pi_1(i)'_{\mathsf{C}})$. Hence, since s_i, s'_i differ in at least one bit, due to the way R_p^* chooses I_i, we have that I_i chosen after query q is distinct from I'_i for the query q' in at least one index. Note also that, by construction $|I_i| = |I'_i| = n$, thus there exist at least one index j_ℓ that is in I'_i but is not in I_i. If both queries q, q' were not aborting, then Sim was able to provide signatures for both sets of indexes, thus Sim

[8] That is, we consider all of the oracle queries made by Sim to R_p^* throughout the simulation and take the subset of those queries which include all the messages belonging to the executions of $\Pi_{2(i-1)n+i+1}$ to Π_{2in+i}. By the scheduling described in Fig. 4.1, such a query defines the i^{th} message of Π_1 that is sent by R^*, (i.e., $\Pi_1(i)_{\mathsf{R}}$.)

provided signature for at least $n+1$ indexes. Recall that, Sim cannot ask the oracle with more indexes that R_p^* would ask in the real world experiment. In particular, for each block of parallel executions, Sim must ask exactly n signatures, according to R_p^* strategy (indeed the strategy of R_p^* is such that R_0^* always asks exactly n openings at each block and rn openings in total, while R_1^* asks for a total of $rn+1$ openings). Thus if R_p^* did not abort in both q, q' this means that Sim was able to provide a total of at least $n+1$ valid signatures to R_p^*, still by asking only n signatures to $\mathcal{O}$, thus Sim has generated a forgery.

Formally the reduction works as follows. We construct a signature-forgery algorithm M that given vk simulates R_p^* and the oracle $\mathcal{O}$ to Sim. M perfectly emulates R_p^* and answers the queries that Sim makes to the oracle by forwarding them to its signing oracle. Note that the emulation is perfect. Now assume that with non-negligible probability there exist $q, q' \in Q_i$ with different messages $\Pi_1(i)_{\mathsf{C}}$, $\Pi_1(i)'_{\mathsf{C}}$ yielding to different set of indexes I_i, I'_i. Since R^* does not abort it must have seen $(\sigma_{j_1}, \ldots, \sigma_{j_n})$ with $j_\ell \in I_i$ and $(\sigma_{j'_1}, \ldots, \sigma_{j'_n})$ with $j'_\ell \in I'_i$. Since there exists at least one index j^* that is not in both sets, R^* gets at least $n+1$ signatures. Recall that for each block Sim is allowed to ask only for n signatures. Thus there exists at least a signature $\sigma^*_{j^*}$ that was not provided by the oracle, hence M outputs $(j^*, \sigma^*_{j^*})$ thus contradicting the security of the signature scheme. □

In Claim 4.3.1 we proved that against adversaries $\{\mathsf{R}_p^*\}_{p\in\{0,1\}}$ any Sim cannot make effective rewinds for the execution Π_1, and thus if Sim exists then it is able to equivocate the first execution without rewinding R_p^*. Since R_p^* in first execution basically follows the procedure of the honest receiver we want to argue that the same strategy used by Sim to equivocate can be adopted by a malicious sender that wants to equivocate but cannot rewind the honest party.

The idea is to construct a malicious sender C^* that runs Sim as subroutine and simulates the same attack of R_1^* except that for the execution Π_1 it forwards the messages to the honest R such that, when Sim asks the oracle for the opening of session 1, C^* replies with the string that it wants to open to R. In particular, it sends the first string and obtains the opening by Sim, then it rewinds Sim and it sends a distinct string, for which it will obtain another valid opening (this is true due to the existence of Sim). Note however that in this reduction we are assuming that Sim asks for the queries only after the execution of Π_1 is completed. Indeed, if Sim queries the oracle for the opening of Π_1 any time before the commitment phase of Π_1 is completed we cannot use Sim as a sub-routine to break binding since Sim could change the transcript of the commitment phase according to the response received from $\mathcal{O}$, and thus in turn if C^* rewinds Sim and changes the string to open, it could obtain a distinct commitment transcript, therefore not violating the binding property. Therefore, before proceeding with the construction of the malicious adversary we need to prove another claim on Sim.

Claim 4.3.2. *In the execution Π_1, except with negligible probability, Sim queries the oracle $\mathcal{O}$ only after receiving the query by $\{\mathsf{R}_p^*\}_{p\in\{0,1\}}$.*

Proof. Since Sim is black-box, it must work for all R^*. In particular Sim must successfully simulate adversaries R_0^* and R_1^*. Adversary R_0^* does not query for the opening the first execution Π_1 (i.e., the probability of opening is $p = 0$), therefore Sim is not required to provide an opening and can simulate Π_1 without interacting with $\mathcal{O}$. Adversary R_1^* always asks to see the opening of Π_1. In this case, Sim could ask for the opening of session 1 to $\mathcal{O}$ at the very beginning of the simulation and thus run the first execution as an honest sender (i.e., with no need of equivocation). The output of Sim would be indistinguishable from the output of C. However, the definition of black-box simulation requires that the same simulation strategy should work for all adversarial strategies. Thus the decision on whether to ask for the opening to $\mathcal{O}$ can be made only after the R_p^* has sent the request of opening. Indeed, if Sim asks the oracle any time *before* it has received the query from R_0^*, then the set of indexes I asked by Sim in the ideal execution is clearly distinguishable since in the real world execution the set I requested by R_0^* does not contain index of Π_1.

The last case to consider is the case that, Sim does not query the oracle for session 1. (Recall that if Sim does not ask the oracle then the malicious sender in the reduction would not be able to exploit Sim to open two distinct strings). Due to the unforgeability of the signature scheme, this case happens with negligible probability. The reduction works as in Claim 4.3.1.

□

Claim 4.3.3. *In the execution Π_1, when dealing with adversary R_1^*, Sim provides a valid opening with all but negligible probability.*

Proof. In Π_1 the adversary R_1^* is playing as the honest receiver, thus always gets a valid opening when interacting with C. By the assumption that Π is SOA-secure under concurrent composition it must holds that also Sim provides a valid opening with all but negligible probability. □

Now we are ready to show the formal construction of the adversary for binding C^* that uses Sim as a sub-routine.

Malicious Sender. C^* playing the binding game, externally interacts with an honest receiver R while internally interacts with Sim. C^* generates a pair $(vk, sk) \xleftarrow{\$} \mathsf{GenSign}(1^n)$, thus defining the set $\mathcal{B}_{sk}$ and gives vk to Sim and R. C^* emulates R_1^* and the oracle $\mathcal{O}$ to Sim for all executions $\Pi_2, \ldots, \Pi_k$. This emulation can be perfectly carried out since it has all the secrets. C^* plays the execution Π_1 by forwarding the messages to the external receiver R. That is, let q be an oracle query from Sim to R_1^* such that R_1^*'s response is the i-th message of execution Π_1. Then, if R_1^* would abort receiving q or any prefix of q, C^* emulates it by aborting. Otherwise, if q is such that R_1^* does not abort but rather replies with the i-th message of Π_1 then C^* extracts the message $(\Pi_1(i)_{\mathsf{C}})$ of the simulator from q, and then, if R has already sent the response $\Pi_1(i)_{\mathsf{R}}$ according to the extracted message, then R_p^* replies this same message to Sim. If R has already sent the answer $\Pi_1(i)_{\mathsf{R}}$ according to another message $\Pi_1'(i-1)_{\mathsf{C}}$ then C^* halts. We call this event as *failure*. Finally, if R did not reply to $\Pi_1(i)_{\mathsf{C}}$ yet, then C^* forwards it to R and stores

the pair $\Pi_1(i)_{\mathsf{C}}$, $\Pi_1(i)_{\mathsf{R}}$. Finally, when Sim makes queries to the oracle $\mathcal{O}$ for a set of indexes I_i, the sender responds via the signatures $\sigma_j = \mathsf{Sign}(j, sk)$ for all $j \in I$.

When Sim queries the oracle for the opening of the execution Π_1, if the commitment phase of the execution with the honest receiver is not completed yet, then C^* aborts and halts. We call this event too as *failure*. Else, C^* provides a string $\alpha_0 \leftarrow \mathsf{Sign}(1, sk, r_0)$ and obtains the opening α_0, w_0 from Sim. Then C^* rewinds Sim up to the point in which it asks the opening for session Π_1 and provides a distinct string $\alpha_1 \leftarrow \mathsf{Sign}(1, sk, r_1)$ to obtain the opening α_1, w_1 from Sim. If Sim never asks the oracle for session Π_1 then C^* aborts and halts. Again we call this event as *failure*.

Finally C^* obtains two openings for the protocol executions Π_1 played with the honest R.

First note that if C^* does not abort then C^* perfectly emulates the attack of R_1^*. Indeed it plays all but the first execution following R_1^* procedure, while for the message of the first execution it forwards the messages receiver by the honest receiver. Again, this is consistent with the strategy of R_1^* since she plays the first execution of Π as an honest receiver.

By Claim 4.3.1 we have that the event *failure* happens only with negligible probability. By Claim 4.3.2 we have that, except with negligible probability, Sim makes the oracle query for the opening of execution Π_1 only after the commitment phase has been completed. By Claim 4.3.3 we have that with all but negligible probability C^* gets two valid openings when interacting with Sim. Thus with high probability C^* provides two valid openings for the commitment phase transcript obtained by playing with R.

Corollary 1. *There exists no bit commitment protocol that is SOA-secure under strong concurrent composition.*

Proof. Toward a contradiction assume that there exists a bit commitment scheme $\Gamma = (\mathsf{C}_\Gamma, \mathsf{R}_\Gamma)$ that is SOA-secure under concurrent composition. Then it is possible to construct a string commitment scheme $\Pi = (\mathsf{C}, \mathsf{R})$ as follows. Let $m = m_0|\ldots|m_n$ be the n-bit message that C wants to commit to. The commitment phase consists of n-parallel executions of the commitments phase of Γ, (i.e., $\langle \mathsf{C}_\Gamma^i(\mathtt{com}, m_i), \mathsf{R}_\Gamma^i(\mathtt{recv})\rangle$ for $i = 1, \ldots, n$). The commitment phase of Π is over when all commitments phases of Γ are successfully completed. The opening phase consists of the parallel execution of the n decommitment phases of Γ (i.e., $\langle \mathsf{C}_\Gamma^i(\mathtt{open}), \mathsf{R}_\Gamma^i(\mathtt{open})\rangle$ for $i = 1, \ldots, n$). Basically, Π consists only of executions of Γ, and since Γ is SOA-secure under concurrent composition, so is Π. □

□

- Chapter 5 -

Round-Optimal Concurrent ZK in the Bare Public Key model

Introduction

The notion of concurrent zero knowledge ($\mathsf{c}\mathcal{ZK}$, for short) introduced in [40] deals with proofs given in asynchronous networks controlled by the adversary.

In [19] Canetti et al. studied the case of an adversary that can reset the prover, forcing it to re-use the same randomness in different executions. They defined as resettable zero knowledge ($\mathsf{r}\mathcal{ZK}$, for short) the security of a proof system against such attacks. Very interestingly, $\mathsf{r}\mathcal{ZK}$ is proved to be stronger than $\mathsf{c}\mathcal{ZK}$.

Motivated by the need of achieving round-efficient $\mathsf{r}\mathcal{ZK}$, in [19] the Bare Public-Key (BPK, for short) model has been introduced, with the goal of relying on a setup assumption that is as close as possible to the standard model. Indeed, round-efficient $\mathsf{c}\mathcal{ZK}$ and $\mathsf{r}\mathcal{ZK}$ are often easy to achieve in other models (e.g., with trusted parameters) that unfortunately are hard to justify in practice.

The BPK model. The sole assumption of the BPK model is that when proofs are played, identities of (polynomially many) verifiers interacting with honest provers are fixed. For instance, identities could be posted to a public directory so that players can download the content of the directory before proofs start. This registration phase is non-interactive, does not involve trusted parties or other assumptions, and can be fully controlled by any adversary. When proofs start, it is assumed that honest provers interact with registered verifiers only.

The BPK model is very close to the standard model, indeed the proof phase does not have any requirement beyond the availability of the directory to all provers, and for each verifier, of a secret key associated to his identity. Moreover, in both phases the adversary has full control of the communication network, and of corrupted players.

Round-optimal $\mathsf{c}\mathcal{ZK}$ in the BPK model from $\mathsf{r}\mathcal{ZK}$. The first constant-round $\mathsf{r}\mathcal{ZK}$ (and thus $\mathsf{c}\mathcal{ZK}$) argument for **NP** in the BPK model has been given in [19]. Then in [73] it is pointed out the subtle separations among soundness notions in the BPK model. Indeed, in contrast to the standard model, the notions of one-time, sequential and concurrent soundness, are distinct in the BPK model. In [73] it is then showed that the proof of [19] is actually sufficient for sequential soundness only. Moreover in [73] it is proved that 4 rounds are necessary for concurrent soundness and finally, they showed a 4-round $\mathsf{r}\mathcal{ZK}$ (and thus $\mathsf{c}\mathcal{ZK}$) argument with sequential soundness. The protocol is "conversation based", i.e., by simply observing the transcript one can compute the output of the verifier. In light of the impossibility proved by [1] (i.e., there exists no 3 round sequentially sound $\mathsf{c}\mathcal{ZK}$ conversation-based argument in the BPK model for non-trivial languages) the above 4-round $\mathsf{r}\mathcal{ZK}$ (and thus $\mathsf{c}\mathcal{ZK}$) argument is round optimal.

Concurrent soundness along with $\mathsf{r}\mathcal{ZK}$ (and thus $\mathsf{c}\mathcal{ZK}$) was achieved in [31], requiring 4 rounds. Further improvements on the required complexity assumptions have been showed in [106] where a 4-round protocol under generic assumptions and an efficient 5-round protocol under number-theoretic assumptions are shown.

We stress that all previously discussed results on constant-round $\mathsf{r}\mathcal{ZK}/\mathsf{c}\mathcal{ZK}$ in the BPK model relied on the assumptions that some cryptographic primitives are secure against sub-exponential time adversaries (i.e., complexity leveraging) and obtained black-box simulation.

Round-optimal $\mathsf{c}\mathcal{ZK}$ in the BPK model under standard assumptions. The question of achieving a constant-round black-box $\mathsf{c}\mathcal{ZK}$ in the BPK model without relying on complexity leveraging has been first addressed in [107] and then in [33]. The protocol of [107] needs 4 rounds and achieves sequential soundness only. The protocol given in [33] also needs 4 rounds and achieves concurrent soundness. A follow up result of [99] showed an efficient transformation that starting from a language admitting a Σ-protocol produces a $\mathsf{c}\mathcal{ZK}$ argument with concurrent soundness needing only 4 rounds and adding only a constant number of modular exponentiations. A more recent result [30] obtains both round optimality and optimal complexity assumptions (i.e., the need of One-way Functions only) for concurrently sound $\mathsf{c}\mathcal{ZK}$. The notion of "knowledge extraction" has been studied in [34] and in [104, 103] where in particular concurrent knowledge extraction (under different formulations) is considered.

All above results achieve $\mathsf{c}\mathcal{ZK}$ and are based on hardness assumptions with respect to polynomial-time adversaries.

Our Contribution

In this work we show subtle problems concerning security proofs of various $\mathsf{c}\mathcal{ZK}$ and $\mathsf{r}\mathcal{ZK}$ arguments in the BPK model [73, 107, 31, 33, 99, 106, 30, 104], including *all* round-optimal constructions published so far.

The source of the problem: parallel execution of different sub-protocols. In order to achieve round efficiency, various known protocols, including all round-optimal protocols, consist in parallel executions of sub-protocols that are useful in different ways in the proofs of soundness and $\mathsf{c}\mathcal{ZK}/\mathsf{r}\mathcal{ZK}$. Roughly speaking, there is always a sub-protocol π_0 where in 3 rounds the verifier is required to use a secret related to its identity. Then there is a 3-round sub-protocol π_1 in which the prover convinces the verifier about the validity of the statement and the simulator can do the same by using knowledge of a secret information obtained by rewinding π_0 (in the current session or in other sessions corresponding to the same identity). To obtain a 4-round protocol[1], π_1 starts during the second round of π_0. Such round combination yields one of the following two cases.

The first case is when the simulator needs the secret information already to compute the first message of π_1 so that such a message can appear in the final transcript of the simulation. In this case when the simulator runs protocol π_1 for the first time with a given identity, it first needs to extract the secret related to such identity, used in π_0 by the verifier. The use of look-ahead threads (i.e., trying to go ahead with a virtual simulation with the purpose of obtaining the required information needed in the main thread of the simulation) would not help here since only a limited polynomial amount of work can be invested for them, and there is always a non-negligible probability that look-ahead threads fail, while still in the main thread the verifier plays the next message. Given the above difficulty, the simulator needs to play a *bad* first round in π_1 so that later, when the needed secret information is extracted from π_0, the simulator can play again such first round of π_1, this time with a *good* message. However, this approach suffers of a problem too. Indeed, aborting the main thread and starting a new thread leads to a detectable deviation in the final transcript that the simulator will output. Indeed, the fact that the simulator gives up with a thread each time it is stuck, and then starts a new one, as we shall see later, skews the distribution of the output of the simulator, since the output will then include with higher probability threads that are "easier" to complete (e.g., where the simulator does not get stuck because new sessions for new identities do not appear). Notice that this issue motivates the simulation strategies adopted in previous work on $\mathsf{c}\mathcal{ZK}$ (e.g., [91, 87]) where the main thread corresponds to the construction of the view that will be given in output, while other threads are started with the sole purpose of extracting secrets useful to complete the main thread. Similar issues concerning the use of a main thread during the whole simulation have been recently considered in [79] when analyzing previous work on selective decommitments.

We now consider the second case where the simulator does not need any secret to compute the first round of π_1. We observe that this approach could hurt the proof of concurrent soundness, when the latter is proved by means of witness extraction[2] from π_1. Indeed, a malicious concurrent prover can exploit the execution of π_0 in a session j, for completing the execution of π_1 in another concurrent session $j' \neq j$ by playing a

[1] Similar discussions hold for some 5-round protocols when π_0 requires 4 rounds.

[2] We note that all constructions of $\mathsf{c}\mathcal{ZK}$ in the BPK model under standard assumptions prove soundness by means of witness extraction.

man-in-the-middle attack such that, when (in the proof of concurrent soundness) one tries to reach a contradiction by extracting the witness from the proof π_1 given in session j', it instead obtains the secret used to run π_0 in session j. Instead, if the secret to be extracted from π_1 is fixed from the very first round of π_1, then one can show that it is either independent from the one used in session j (this happens when the secret is used in π_0 of session j after the first round of π_1 in session j' is played, and the secret used by the verifier can not be predicted with non-negligible probability), or is dependent but not affected by the rewind of session j' (this happens when the secret is used in π_0 of session j before the first round of π_1 in session j' is played).

The use of the secret in the last round of π_1 only, could instead be helpful in the following three cases: I) when one is interested in $\mathsf{r}\mathcal{ZK}$ since in this case soundness is proved through a reduction based on complexity leveraging (no need for rewinding); II) when $\mathsf{c}\mathcal{ZK}$ with sequential soundness only is desired; III) when the secret needed by the simulator when running π_1 in a session j' is different from the witness used by the verifier in the execution of π_0 in other sessions. In these three cases the above discussion does not necessarily apply. Indeed some proposed round-optimal protocols that fall in one of such cases, might still be secure (see discussion in Section **??**), even though their security proofs seem to ignore at least in part the problems that we are pointing out.

Because of the above case I), we believe that achieving 4-round $\mathsf{c}\mathcal{ZK}$ with concurrent soundness in the BPK model under standard assumptions is definitively harder than obtaining 4-round $\mathsf{r}\mathcal{ZK}$ with concurrent soundness in the BPK model through complexity leveraging. Therefore, in this thesis we focus on achieving $\Pi_{\mathsf{c}\mathcal{ZK}}$ and this will require a new technique.

We stress that in all previous constructions, one could obtain a different protocol that satisfies the desired soundness and zero-knowledge properties by simply running π_0 and π_1 sequentially. Indeed, in this case the simulator can complete π_0 in the main thread, then run the extractor in another thread, and finally continue the main thread running π_1 having the secret information. We also stress that all papers that we revisit in this work, achieved also other results that are not affected by our analysis when round optimality is not desired.

We finally note that we did not investigate other round-efficient results in variations of the BPK model [72, 108, 32], and other results in the BPK model that do not focus on optimal round complexity [78, 105, 24].

New techniques for round-optimal $\mathsf{c}\mathcal{ZK}$ in the BPK model. In this thesis we show a protocol and a security proof that close the gap in between lower and upper bounds for the round complexity of concurrently sound $\mathsf{c}\mathcal{ZK}$ in the BPK model under standard assumptions. The result is achieved by using a new technique where in addition to the (permanent) secret associated to the identity of the verifier, there is a *temporary* secret per session, that enables the simulator to proceed in two modes as follows. Knowledge of the permanent secret of the verifier allows the simulator to proceed in straight-line in the main thread in sessions started after the extraction of the permanent secret. Knowledge of the temporary secret allows the simulator to solve the

sessions started before the extraction of the permanent secret, by launching rewinding threads but without changing the main thread. The temporary key is extracted through rewinding threads, and it is used only when computing the last prover message of a session of the main thread, i.e., only after the extraction has been completed. This allows to keep the main thread unchanged. In the rewinding threads the simulator is always straight-line. We implement both the permanent and the temporary keys by means of trapdoor commitments. The proof of $\mathsf{c}\mathcal{ZK}$ will be tricky since it requires the synergy of the two above simulation modes. In our case the number of extraction procedures required to carry on the simulation is not bounded by the number of identities registered in the directory (in contrast with the main technique used in the past in the BPK model), but by the number of sessions. The proof of concurrent soundness also requires special attention. Indeed while the interplay of temporary and permanent secrets helps the simulator, it could also be exploited by the malicious prover.

Finally, we show that $\Pi_{\mathsf{c}\mathcal{ZK}}$ admits a transformation that starting from any language admitting a perfect Σ-protocol, produces round-optimal concurrently-sound $\mathsf{c}\mathcal{ZK}$ protocol $\bar{\Pi}_{\mathsf{c}\mathcal{ZK}}$. Such transformation requires a constant number of modular exponentiations only, and $\bar{\Pi}_{\mathsf{c}\mathcal{ZK}}$ is secure under standard number-theoretic assumptions.

It is plausible that motivated by different purposes one can get a more general construction or a construction with better efficiency, assumptions and additional security, but this is out of the scope of this work.

The chapter is organized as follows. In Section 5.1 we provide the formal definition of argument system in the BPK model. In Section 5.2 we show an attack that jeopardizes the security of all previous constructions of $\mathsf{c}\mathcal{ZK}$ in the BPK model. In Section 5.3 we provide our round-optimal construction which does not suffer of such attack. Finally, in Section **??** we show that our construction can be efficiently instantiated using the Σ-protocols shown in Section 1.2.1.

5.1 The Bare Public-Key Model

In this paragraph we follow the definitions provided by [73, 1]. The BPK model assumes that: 1) there exists a polynomial-sized collection of records associating identities of the verifiers with public keys in a public file F; 2) an (honest) prover is an interactive deterministic polynomial-time algorithm that takes as input a security parameter 1^n, the public file F, an n-bit string x, such that $x \in L$ where L is an **NP**-language, an auxiliary input w, a reference to an entry of F and a random tape ω; 3) an (honest) verifier $\mathcal{V}$ is an interactive deterministic polynomial-time algorithm that works in the following two stages: **key-generation phase**, on input a security parameter 1^n and a random tape ρ, $\mathcal{V}$ generates a pair (PK, SK) and stores its identity associated with PK in an entry of the file F; **proof phase**, $\mathcal{V}$ takes as input the secret SK associated to its identity, a statement $x \in L$ and a random string, and interacts with a prover and when reaching the end of the interaction without aborting, it outputs "accept" or "reject"; 4) the first interaction of a prover and a verifier starts after all verifiers have completed

their key-generation phase.

Procedure Normal-Interaction

1. Run the key-generation phase of $\mathcal{V}$ on input 1^n and a random string r to obtain PK, SK;
2. Pick any id, and let F be a public file that contains the record (id, PK);
3. Pick strings ω and ρ at random and run $\mathcal{P}$ on inputs $(1^n, x, w, id, \omega)$, and the proof phase of $\mathcal{V}$ on inputs SK, x, ρ letting them interact.

Definition 20. *A pair $(\mathcal{P}, \mathcal{V})$ is complete in the* BPK *model for an* **NP***-language L if for all n-bit strings $x \in L$ and their* **NP***-witnesses w, in an execution of Normal-Interaction $\mathcal{V}$ outputs "accept" with probability* 1*.*

Malicious provers in the BPK model. Let s be a positive polynomial and $\mathcal{P}^*$ be a probabilistic polynomial-time algorithm that takes as first input 1^n.

An s-sequential malicious prover $\mathcal{P}^*$ runs in at most $s(n)$ stages, so that:

1. In stage 1, $\mathcal{P}^*$ receives a public key PK and outputs a string x_1 of length n.
2. In every even stage, $\mathcal{P}^*$ starts in the final configuration of the previous stage and performs a single interactive protocol: it outputs outgoing messages and receives incoming messages (the machine with which it performs the interactive protocol will be specified below, in the definition of sequential soundness). It can choose to abort an even stage at any point and move on to the next stage by outputting a special message.
3. In every odd stage $i > 1$, $\mathcal{P}^*$ starts in the final configuration of the previous stage and outputs a string x_i of length n.

Procedure Sequential-Attack

1. Run the key-generation stage of $\mathcal{V}$ on input 1^n and a random string r to obtain PK, SK.
2. Run first stage of $\mathcal{P}^*$ on inputs 1^n, random string ω and PK to obtain an n-bit string x_1.
3. For i ranging from 1 to $s(n)/2$:
    1. Select a random string ρ_i.
    2. Run the $2i$-th stage of $\mathcal{P}^*$ letting it interact with the verification stage of $\mathcal{V}$ with input SK, x_i, ρ_i.
    3. Run the $(2i+1)$-th stage of $\mathcal{P}^*$ to obtain an n-bit string x_{i+1}.

Definition 21. *A complete pair $(\mathcal{P}, \mathcal{V})$ satisfies sequential soundness for a language L if for all positive polynomials s, for all s-sequential malicious provers $\mathcal{P}^*$, the probability that in an execution of Sequential-Attack, there exists i such that $1 \leq i \leq s(n), x_i \notin L$, and $\mathcal{V}$ outputs "accept x_i" is negligible in n.*

An s-concurrent malicious prover $\mathcal{P}^*$, on inputs 1^n and PK, performs at most $s(n)$ interactive protocols as follows:

1. If $\mathcal{P}^*$ is already running $i-1$ interactive protocols $1 \leq i-1 < s(n)$, it can output a special message "Start x_i" where x_i is a string of length n.
2. At any point it can output a message for any of its (at most $s(n)$) interactive protocols (the protocol is unambiguously identified in the outgoing message). It then immediately receives the party's response and continues.

Procedure Concurrent-Attack

1. Run the key-generation stage of $\mathcal{V}$ on input 1^n and a random string r to obtain PK, SK.
2. Run $\mathcal{P}^*$ on inputs 1^n, random string ω and PK.
3. Whenever $\mathcal{P}^*$ outputs "Start x_i" select a fresh random string ρ_i and let the i-th machine with which $\mathcal{P}^*$ interacts be the verification stage of $\mathcal{V}$ on inputs SK, x_i, ρ_i.

Definition 22. *A complete pair $(\mathcal{P}, \mathcal{V})$ satisfies concurrent soundness for a language L if for all positive polynomials s, for all s-concurrent malicious provers $\mathcal{P}^*$, the probability that in an execution of Concurrent-Attack, $\mathcal{V}$ ever outputs "accept x" for $x \notin L$ is negligible in n.*

Definition 23. *Let $(\mathcal{P}, \mathcal{V})$ be an interactive argument system in the* BPK *model for a language L. We say that a probabilistic polynomial-time adversarial verifier $\mathcal{V}^*$ is a* **concurrent adversary in the** BPK **model** *if on input polynomially many values $\bar{x} = x_1, \ldots, x_{\mathtt{poly}(n)}$, it first generates the public file F with $\mathtt{poly}(n)$ public keys and then concurrently interacts with $\mathtt{poly}(n)$ number of independent copies of $\mathcal{P}$ (each with a valid witness for the statement), with common input $\bar{x}$ and without any restrictions over the scheduling of the messages in the different interactions with $\mathcal{P}$. Moreover we say that the transcript of such a concurrent interaction consists of a vector of instances $\bar{x}$ and the sequence of prover and verifier messages exchanged during the interaction. We refer to $\mathtt{view}^{\mathcal{P}}_{\mathcal{V}^*}(\bar{x})$ as the random variable describing the content of the random tape of $\mathcal{V}^*$ and the transcript of the concurrent interactions between $\mathcal{P}$ and $\mathcal{V}^*$.*

Definition 24. *Let $(\mathcal{P}, \mathcal{V})$ be an interactive argument system for a language L in the* BPK *model. We say that $(\mathcal{P}, \mathcal{V})$ is black-box computational (resp., statistical, perfect)* **concurrent zero knowledge** *(c$\mathcal{ZK}$ for short) if there exists a probabilistic polynomial-time algorithm* Sim *such that for each polynomial-time concurrent adversary $\mathcal{V}^*$, let $\mathsf{Sim}^{\mathcal{V}^*}(\bar{x})$ be the output of* Sim *on input $\bar{x}$ and black-box access to $\mathcal{V}^*$, then if $x_1, \ldots, x_{\mathtt{poly}(n)} \in L$, the ensembles $\{\mathtt{view}^{\mathcal{P}}_{\mathcal{V}^*}(\bar{x})\}$ and $\{\mathsf{Sim}^{\mathcal{V}^*}(\bar{x})\}$ are computationally (resp., statistically, perfectly) indistinguishable.*

5.2 Issues in Security Proofs of Previous Results

In this section we show an issue that applies to the security proofs of all previous work. Towards this goal, we organize this section in two parts.

In the first part, we consider a generic protocol Π_{weak}. We describe the simulation strategy that is typically used in the security proof of all previous work (or at least the most plausible interpretation of it), and we show that, using such simulator, one can

(wrongly) prove that Π_{weak} is $\mathsf{c}\mathcal{ZK}$ in the BPK model. Then we show an attack to this simulation strategy. Namely, we show that there exists a malicious verifier $\mathcal{V}^*$, for which the output of such simulator is clearly distinguishable from the output of a real-world execution. This directly implies that, when security is proven using such simulation strategy, the protocol is not actually secure. We then investigate whether there exists a different simulation strategy, and show that, due to the protocol's structure of Π_{weak}, designing any simulator seems to be problematic.

In the second part we briefly go over some of the previous work. We consider each protocol and simulation strategy, and observe that under minor variations, protocols and simulators follow the same protocol's structure and the same strategy discussed for protocol Π_{weak}. Therefore, any alternative proof of security for such protocols seems problematic to design as well.

In the next section instead we will cover previous work [107, 106, 104, 103, 30] that, although using the same simulation strategy described for Π_{weak}, do not actually follow the same protocol structure (or they can be easily modified to not follow it). Therefore, for such protocols an alternative simulation strategy may exist.

5.2.1 A Generic Protocol Π_{weak}

Consider the following protocol. The public identity of the verifier is a pair (y_0, y_1) of evaluations of a OWF for which the verifier $\mathcal{V}$ knows only one of the pre-images, denoted by sk. The verifier $\mathcal{V}$ starts by proving knowledge of sk using a 3-round WI proof of knowledge protocol π_0 where the witness sk is used in the third round. The prover $\mathcal{P}$ **first** commits to a string, and then proves that the commitment hides either the witness for the theorem $x \in L$ or one of the pre-images of (y_0, y_1), using a Σ-protocol. We denote by π_1 the whole protocol run by $\mathcal{P}$ (i.e., the commitment of the witness and the Σ-protocol). To achieve round optimality, protocol π_1 starts on the second round of π_0. Hence, the commitment of the witness is sent already in the second round. The protocol is depicted in Figure 5.1.

We now (wrongly) argue that this protocol is concurrent $\mathcal{ZK}$ in the BPK model borrowing the simulation strategy used in literature.

A typical simulator Sim discussed in literature would work as follows. It runs the extractor of protocol π_0 to obtain the secret sk. Once this secret is obtained, the identity associated to such secret is solved. Then it commits to sk and runs in straight-line protocol π_1 using sk as witness. However, there is an ambiguity here. The simulator should commit to sk already in the second round. But, sk must be extracted first. In order to extract sk from π_0, Sim should run the first three rounds, in particular it has to compute the commitment of sk in the second round (otherwise it will not have the witness to complete π_1), without knowing sk yet. We can give two interpretations to this ambiguity.

Case 1. The first interpretation is to assume that the extraction of sk is performed in a look-ahead thread that is played before the main thread (where the simulator computes the actual messages to be given in output). In this case, notice that the proof of knowledge of sk could be completed by $\mathcal{V}^*$ with some probability p unknown to

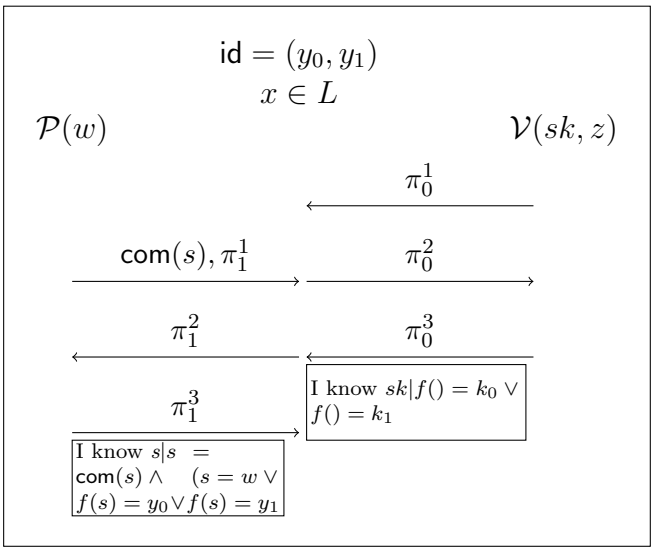

Figure 5.1: *Protocol* Π_{weak}.

Sim (Sim is black-box, and there can be different adversarial verifiers using different values for p, and some of them can be negligible). Therefore, since the attempt of Sim to extract sk can not be (in order to have an expected polynomial time simulation) unlimited in time, Sim must give up if after some polynomial effort sk has not been extracted. When such a look-ahead thread is aborted, Sim continues the main thread. At this point, it can happen with non-negligible probability p (since Sim stopped after a polynomial number of attempts) that $\mathcal{V}^*$ completes the proof of knowledge of sk. Since in this case Sim has already played the second round, the string committed is very likely not a pre-image for y_0 or y_1, therefore Sim can not conclude protocol π_1. If one gives in output such a failure, then the transcript of the simulation would be easily distinguishable since in the view of the $\mathcal{V}^*$ when playing with $\mathcal{P}$ there is no failure from $\mathcal{P}$. Therefore, in this case Sim needs to abort this main thread and start a new one, having now sk as input. This strategy corresponds to Case 2.

Case 2. The second interpretation consists in assuming that Sim runs the proof of knowledge π_0, playing a bad commitment in the second round (a bad commitment is one that does not allow the simulator to complete π_1 in straight-line). Once the verifier completes π_0, Sim runs the extractor of the proof of knowledge π_0 and in time roughly $\mathtt{poly}(n)/p$, where p is the probability that $\mathcal{V}^*$ completes π_0, it obtains the secret associated to the identity of $\mathcal{V}^*$, the pre-image sk. Then Sim rewinds the verifier and starts the proof phase of the simulation from scratch, without changing the key generation phase. Using knowledge of the secret sk, Sim can complete all sessions that correspond to that solved identity in straight-line. Indeed, for such identities Sim computes a good commitment that allows to complete protocol π_1 using sk as witness.

This approach is often used in literature in the BPK model and consists in dividing

the simulation in phases. In each phase the simulator runs in straight-line using the secrets extracted so far. When Sim is stuck with an identity for which it does not know the secret, it extracts the secret from π_0, solves the identity, and starts a new phases with the knowledge of one more secret. Since the number of identities is polynomial, at some point there will be a phase in which the simulator resolves all the sessions in straight-line. The simulator gives in output the transcript obtained from the last phase.

Using the simulation strategy shown in Case 2 one can (wrongly) prove that Π_{weak} is $\mathsf{c}\mathcal{ZK}$ in the BPK model. Let us denote such simulator as $\mathsf{Sim}_{\mathsf{weak}}$.

An Attack to $\mathsf{Sim}_{\mathsf{weak}}$

Consider the following simple attack. The adversary $\mathcal{V}^*$ runs two nested sessions, corresponding to two different identities and such that in each session the third round is played with probability 1/2, adaptively to the transcript obtained so far (e.g., this can be easily done by assuming that the coins used for such a probability are taken from the output of a Pseudo-Random Function (PRF) on input the transcript so far and a seed hardwired in $\mathcal{V}^*$). The nesting is performed by including the whole execution of the second session in between the second and third round of the first session. A pictorial representation of the nested sessions is provided in Figure 5.2. The transcript of the real game with probability 1/4 includes the two sessions both aborted.

Instead, the output of $\mathsf{Sim}_{\mathsf{weak}}$ will be computed as follows. With probability 1/4, $\mathcal{V}^*$ aborts in both sessions. In this case the simulation is straight-line, and outputs two aborts. With probability 1/4, $\mathcal{V}^*$ does not abort the first session (probability 1/2) and it aborts the second session (probability 1/2). In this case, $\mathsf{Sim}_{\mathsf{weak}}$ runs the extractor for π_0 played in the first session and obtains the secret pre-image of the first identity. Once such identity is solved, $\mathsf{Sim}_{\mathsf{weak}}$ starts a new phase, this time having the secret key of the first identity as input. We stress that $\mathsf{Sim}_{\mathsf{weak}}$ is forced to start a new phase since it can not continue the previous execution. The reason is that the commitment sent in the second round of the first session, was not the commitment of the secret sk, that instead was extracted only after the third round of the protocol has already been played. Now notice that in this new phase (that was started with probability 1/4), $\mathsf{Sim}_{\mathsf{weak}}$ will play a new *good* second round in the first session, thus changing the transcript seen by $\mathcal{V}^*$, and therefore the coins used to decide whether to abort or not, will be computationally independent. Hence, $\mathcal{V}^*$ can abort both sessions again with probability 1/4. In this case $\mathsf{Sim}_{\mathsf{weak}}$ outputs two aborts again. Thus, since the new phase started with probability 1/4, and in this new phase $\mathcal{V}^*$ aborts two sessions with probability 1/4, we have that $\mathsf{Sim}_{\mathsf{weak}}$ outputs two abort sessions with probability $\frac{1}{16} = \frac{1}{4} \cdot \frac{1}{4}$.

Therefore we have that the output of the simulator contains both session aborted with probability at least $\frac{5}{16} = \frac{1}{4} + \frac{1}{16}$. Given that in the real game this probability is only 1/4, we have that the output of the simulator is trivially distinguishable. For simplicity in the above analysis we have ignored the fact that $\mathcal{V}^*$ uses a PRF instead of independent coins, since this produces only a negligible deviation to our analysis, and moreover the PRF could be replaced by a t-wise independent hash functions, for a sufficiently large t.

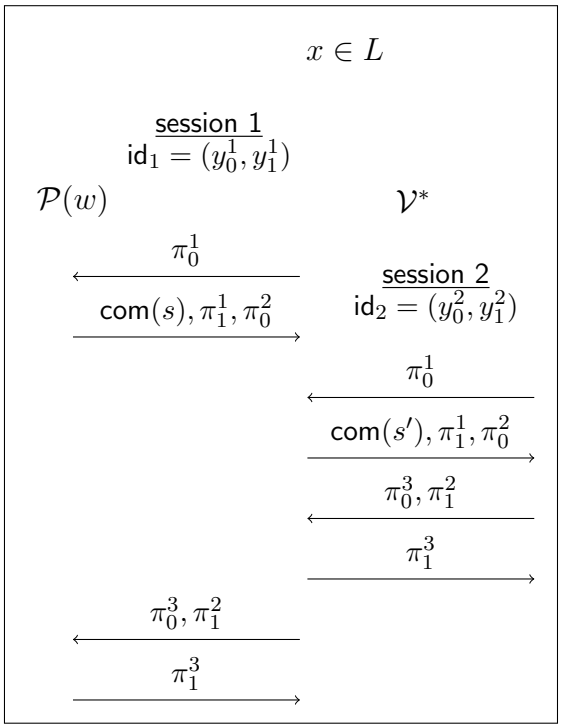

Figure 5.2: *The nested attack of* $\mathcal{V}^*$.

Alternative Simulation? Given the above explicit attack, one might wonder if a different simulator or a different interpretation of Sim can be used to prove the $\mathsf{c}\mathcal{ZK}$ property of Π_{weak}. Indeed, the above attack is certainly addressable with a slightly more sophisticated ad-hoc simulator. However other more sophisticated attacks can easily hurt the new simulation strategy, as in a cat and mouse game where given an adversary one can find a valid simulator for it; but given the valid simulator for that adversary one can find another adversary that requires another simulator. It is not clear at all whether one can finally design a simulator that works against any adversary, as required by the definition of black-box zero knowledge.

The main difficulty in designing a successful simulation strategy seems due to the protocol's structure. The reason is that the secret to be extracted is needed already by the simulator in the second round. However, to extract the secret, Sim needs to play at least the first three rounds, so to complete π_0. Thus, there seems to be no choice for the simulator but playing a *bad* second round. Then, once the secret is extracted, in order to complete π_1, Sim has to rewind $\mathcal{V}^*$ and play a *good* second round, therefore changing the transcript and skewing the distribution given in output. As we shall see later, many protocols in literature follow such structure, and require that the secret of the verifier is used already in the second round.

We stress that, such problem arises only when round optimality is aimed. Indeed, if protocol π_0 and π_1 are played sequentially, then the simulator can first extract the secret and then compute the commitment, without having to change the transcript played so far. In general, all the round optimal protocols that suffer of the issue that we point out, are instead secure when the sub-protocols of $\mathcal{V}$ and $\mathcal{P}$ are run sequentially (instead of concurrently).

The above difficulties are not an issue when considering the simulators for concurrent zero knowledge [91, 65, 87] that indeed use the following strategy: the simulator starts a main thread that is updated with new messages exchanged with $\mathcal{V}^*$; other threads are started only to allow the main thread to proceed successfully, but no thread ever replaces the main thread. This is a well understood strategy that we will also use in our constructions. It however will require a new technique to design a protocol where new threads can help the execution of the main thread (this is precisely the problem of some of previous constructions). The strategy of [91] is actually based on starting look-ahead threads, and the large round complexity tolerates failures of several look-ahead threads as long as there is at least one successful look-ahead thread per session.

5.2.2 Same Attack Applied Everywhere

Here we briefly go (in chronological order) over round-optimal protocols shown in literature [73, 107, 31, 33, 99, 106, 30, 104]. We show that, under minor variations, all proposed simulators follow the strategy of $\mathsf{Sim}_{\mathsf{weak}}$, therefore, they all suffer of the same attack discussed above. We also point out that they follow the protocol's structure of Π_{weak} and therefore, also for them, it seems problematic to come up with any alternative simulation strategy.

The Case of Π_{MR} [73] Micali and Reyzin showed in [73] a 4-round $\mathsf{r}\mathcal{ZK}$ (and thus $\mathsf{c}\mathcal{ZK}$) argument with sequential soundness in the BPK model. The identity of the verifier $\mathcal{V}$ is a public-key pk for a semantically secure encryption scheme, and the secret key sk is the corresponding private key. In the first round, $\mathcal{V}$ sends an encryption c under pk of a random string $\sigma_{\mathcal{V}}$. The prover $\mathcal{P}$ sends in the second round a random string $\sigma_{\mathcal{P}}$. In the third round $\mathcal{V}$ sends $\sigma_{\mathcal{V}}$ and the randomness used to compute c. Moreover in these first 3 rounds, $\mathcal{V}$ proves to $\mathcal{P}$ knowledge of sk using Blum's protocol for Hamiltonicity [12]. In the fourth round $\mathcal{P}$ sends a non-interactive zero knowledge (NIZK, for short) proof [42] on string $\sigma = \sigma_{\mathcal{V}} \oplus \sigma_{\mathcal{P}}$ proving that $x \in L$.

The simulator of Π_{MR} does not achieve $\mathsf{c}\mathcal{ZK}$. The simulator Sim discussed in [73] (see also [90]) for Π_{MR} goes as follows. It runs the extractor associated to the proof of knowledge, therefore obtaining sk. Then, it can run in straight-line since the encryption c of $\sigma_{\mathcal{V}}$ can be decrypted using sk, and thus Sim can choose $\sigma_{\mathcal{P}}$ so that the resulting σ corresponds to the fake random string generated by the NIZK simulator. Then Sim can complete the protocol running in the 4th round the NIZK simulator. Such simulator can have two interpretations as we have seen in Section 5.2.1. Thus, the same analysis shown in Section 5.2.1 holds. Furthermore, here Sim needs to decrypt the message c before sending $\sigma_{\mathcal{P}}$. Thus, knowledge of the secret is required already in the second round. Hence, Π_{MR} follows the same protocol structure of Π_{weak}. Consequently, there is no evidence that Π_{MR} be $\mathsf{c}\mathcal{ZK}$.

The Case of Π_{DPV} [31] In [31], Di Crescenzo, Persiano and Visconti presented the first constant-round concurrently sound $\mathsf{r}\mathcal{ZK}$ (and thus $\mathsf{c}\mathcal{ZK}$) argument system Π_{DPV} for **NP** in the BPK model. The protocol only requires 4 rounds, and thus is round optimal.

The main technique used in [31] to upgrade the sequential soundness achieved in [73] consists in the use of a puzzle sent from the prover to the verifier in the second round of the protocol. The puzzle is implemented through a one-way-permutation and can be inverted running in super-polynomial time, but still below the assumed hardness of all other involved primitives. The proof of knowledge given by the verifier actually proves either knowledge of the private-key of the encryption scheme, or of the solution of the puzzle. The remaining part of the protocol is similar to the one of Π_{MR}.

The simulator of Π_{DPV} does not achieve $\mathsf{c}\mathcal{ZK}$. As discussed in [31] (see page 248), the simulation goes through phases where each phase is straight-line until the simulator is stuck. Then one more secret key is extracted and the simulation starts a new thread trying to complete it, this time using one more secret key. As discussed in Section 5.2.1 since the simulator does not keep the main thread, more easily completes threads when the verifier decides to abort many sessions (there are less chances that the simulator would be stuck in those threads). The attack shown for $\mathsf{Sim}_{\mathsf{weak}}$ breaks analogously also the simulation given in Π_{DPV}. Consequently there is no evidence that Π_{DPV} be $\mathsf{c}\mathcal{ZK}$.

The Case of Π_{DV} [33] In [33], Di Crescenzo and Visconti presented a 4 round $\mathsf{c}\mathcal{ZK}$ argument for **NP** in the BPK model under standard assumptions. In their protocol Π_{DV}, π_0 is a WI proof of knowledge of one over two trapdoors for a trapdoor commitment scheme. In the second round $\mathcal{P}$ commits to a random string, using the trapdoor commitment scheme, and concurrently $\mathcal{P}$ proves theorem $x \in L$ using a Σ-protocol (played in rounds 2,3,4) where the challenge is computed as the xor of the challenge picked by the verifier, and a challenge that was committed by the prover through the trapdoor commitment played in round 2. The simulator extracts the trapdoor from π_0 and equivocates the commitment sent in the second round, adaptively on the challenge played by $\mathcal{V}$. However, the key point is that, the trapdoor commitment scheme used in their construction, is such that a commitment can be equivocated only if the trapdoor was used already in the commitment phase. This property is fundamental in their proof of soundness. Therefore, again, in order to equivocate, Sim needs to know the trapdoor already in the second round.

The simulator of Π_{DV} does not achieve $\mathsf{c}\mathcal{ZK}$. The proof of $\mathsf{c}\mathcal{ZK}$ of Π_{DV} (see page 824 of [33]) requires that the simulator computes the commitment sent in the second round, without having extracted the trapdoor yet. Therefore the simulator is stuck when it needs to play the fourth round (after getting the third round from the verifier) since it can not equivocate anymore (in order to equivocate, knowledge of the trapdoor is needed already when the commitment is computed). The suggestion given in [33] to carry on the simulation consists in extracting the secret key so that later on, in a next thread, all commitments computed by the simulator will be equivocal. Again, as discussed before, the fact that the simulator does not keep the main thread and keeps starting new threads until the execution with the verifier is completed without being stuck is clearly distinguishable. Consequently, there is no evidence that Π_{DV} be $\mathsf{c}\mathcal{ZK}$.

The Case of Π_V [99]

In [99], Visconti showed an efficient transformation for obtaining a 4 round $\mathsf{c}\mathcal{ZK}$ argument in the BPK model starting from any language admitting a Σ-protocol and only adding a constant number of modular exponentiations.

The protocol is based on a proof of knowledge of one over two discrete logarithms (the two possible secret keys) given by $\mathcal{V}$ in the first 3 rounds. Then in the second round the prover commits through a variation of the perfectly binding commitment due to Micciancio and Petrank [74] to one of two secret keys (this is possible also without knowing the secret keys, just relying on multiplications and exponentiations in the considered groups). In rounds 2, 3 and 4, the prover proves that the above commitment corresponds to one of the two secrets. Then it essentially gives a Σ-protocol for $x \in L$ where knowledge of the decommitment of the above commitment allows one to complete the Σ-protocol without knowing the witness.

The simulator for $\mathsf{c}\mathcal{ZK}$ of Π_V does not achieve $\mathsf{c}\mathcal{ZK}$. The simulator for $\mathsf{c}\mathcal{ZK}$ of Π_V discussed in [99], is based on the capability of equivocating concurrently the commitments played in round 2. In order to be able to equivocate the simulator is required to extract the trapdoor (i.e., a discrete logarithm) from the proof given by the verifier in the first 3 rounds (see pages 29-30 of [99]). This means that in order to reach the third round the simulator is required to commit in the second round without having the trapdoor. Although, the scheme allows to commit to one of the two possible trapdoors without having them, the simulator will select the wrong trapdoor with probability 1/2, i.e., it selects the trapdoor that is not extracted from the proof of knowledge given by $\mathcal{V}$. Therefore, Sim can not equivocate the commitment sent in the 2nd round. Sim is therefore stuck, and it needs to start a new thread where it commits to the extracted trapdoor. However, as discussed previously, the fact that the simulator starts new threads when it is stuck is clearly attackable by malicious concurrent verifiers as the one shown for $\mathsf{Sim}_{\mathsf{weak}}$. Consequently, there is no proof that Π_V be $\mathsf{c}\mathcal{ZK}$.

The Case of Π_{DV2} [34] In [34] Di Crescenzo and Visconti focus on the notion of knowledge extraction in the BPK model, which we do not consider in this work. However, they also claim the existence of a round-optimal concurrently sound $\mathsf{c}\mathcal{ZK}$ argument system Π_{DV2} in the BPK model under more general assumptions than the ones used in [33] for Π_{DV}. The protocol actually corresponds to the one of [33] but the underlying primitives are implemented under general complexity-theoretic assumptions instead of number-theoretic assumptions. Since both protocols have the same structure, the same issues discussed for Π_{DV} apply to Π_{DV2} too.

To conclude, in this section we have proved via a counterexample that the simulation strategy proposed in literature leads to a distinguishable output. We also argued that for some of the protocols, finding alternative simulation strategy seems hard, due to the protocol structure.

5.2.3 Replacing Simulation in Phases by Threads

We now discuss four previous protocols that besides the issues in the proposed simulation strategies discussed above, seem (in some cases with some fixes) still to be able to admit a simulation strategy based on maintaining a main thread. We stress that later we will show a new technique based on the use of temporary keys along with permanent keys so that the simulator works in two modes that allow it to stick with the main thread. As far as we know, our technique was never used in previous papers. Protocols below when using a different simulation strategy (in some cases, our new simulation strategy) can potentially achieve some of the results that we will achieve in the next section. We did not go through details of the proofs of such (in some cases, fixed) 4 protocols. We do not claim their security and here we only explain how such protocols and their (distinguishable) simulations in phases could potentially be adjusted in light of our results and techniques.

The Case of Π_Z [107]. A 4-round conversation-based $\mathsf{c}\mathcal{ZK}$ argument enjoying sequential soundness only is shown in [107]. While the security proof still relies on the use of a simulator that works in phases, we notice that a different simulator based on keeping a main thread could be used instead. The reason is that the secret information is needed by the simulator only in the third round of π_1 (see our discussion in Section **??**) and, since the achieved result is only sequentially sound, there is no concurrent attack to soundness to take care of.

The Case of Π_{YZ} [106]. In [106], Yung and Zhao showed protocols Π_{YZ} and $\bar{\Pi}_{YZ}$ that are respectively a 4-round concurrently sound $\mathsf{r}\mathcal{ZK}$ argument in the BPK model under general complexity-theoretic assumptions and an efficient 5-round concurrently sound $\mathsf{r}\mathcal{ZK}$ argument under number theoretic assumptions. Both protocols use complexity leveraging and we will now concentrate on Π_{YZ} since the analysis extends also to $\bar{\Pi}_{YZ}$ with one more round.

Π_{YZ} consists of 3 sub-protocols played in parallel. In the first three rounds the verifier, using a special Σ-protocol Σ^{fls}, gives a proof of knowledge of its secret key sk or of a solution of a puzzle. The puzzle was sent by the prover during the second round, and Σ^{fls} is such that knowledge of the theorem (and therefore of the witness) is not required in the first round. The prover gives a resettable WI proof (i.e., the verifier commits to the challenge in the first round) in rounds 2, 3 and 4 where it proves that $x \in L$ or it knows sk. Since black-box extraction of the witness (necessary for the proof of concurrent soundness) is not allowed in the resetting verifier setting, they enforce the extraction using complexity leveraging as follows. The challenge is committed through a trapdoor commitment scheme with a 2-round decommitment phase, where the trapdoor, that is needed only in the opening, corresponds to the solution of the puzzle sent by the prover. Therefore there exists a sub-exponential time extractor that can find the solution of the puzzle, open the commitment in multiple ways and thus extract the actual witness of the prover. This proof of concurrent soundness falls down when one would like to use standard hardness assumptions only (e.g., to prove $\mathsf{c}\mathcal{ZK}$ under standard assumptions). The technical difficulty of implementing efficiently Σ^{fls} is solved by requiring the prover to send the puzzle in a first round, so that an OR composition of Σ-protocols can be used, therefore obtaining a 5-round protocol $\bar{\Pi}_{YZ}$.

As discussed in [106] (see page 136), the simulator runs in different phases, trying in each phase to complete the simulation, but in case it can not, it obtains a new secret key and starts a new phase, with new randomness. This approach, as previously discussed, makes the output of the simulator distinguishable when playing with some specific adversarial concurrent verifiers. However, we notice that in this case an alternative simulation strategy could be possible. Indeed, when the simulator starts the main thread and gets stuck, it does not actually need to abort it, but instead can start a new thread just to get the secret information to complete the main thread. The reason why this can be possible here (in contrast to previous protocols), is that the simulator needs the secret of the verifier only when it plays the last message of the protocol, therefore it can always perform the extraction (in a new thread) before being stuck. However, as discussed in

the introduction, playing the extracted secret only in the last round exposes the protocol to concurrent soundness attacks. In the very specific case of $\mathsf{r}\mathcal{ZK}$, since soundness is proved through complexity leveraging, the proof of soundness could go through.

We stress that it is not clear at all how to prove concurrent soundness by only relying on standard assumptions (i.e., without complexity leveraging). We will later show our construction that is based on standard assumptions.

The Case of Π_{YYZ} [104, 103]. In the concurrently sound $\mathsf{c}\mathcal{ZK}$ protocol presented in [104, 103], the simulator is required to commit in the second round to one of the two secret keys of the verifier. This must be done before the verifier completes its proof of knowledge of one of her secret keys. It is immediate to see that precisely as we discussed above, this requires the simulator to try to complete the simulation using new phases (see page 24 of [103]). Therefore the same attacks showed before can be mounted against this simulator too.

In Section 6.2 of [103] an update of the protocol yielding round optimality is suggested. The update consists in replacing a strong WI proof with a 4-round zero-knowledge argument of knowledge argument due to Feige and Shamir [43] (FSZK, for short), which would also subsume the proof of knowledge given by the verifier. However, in the same section it is then observed that such update hurts the concurrent soundness of their scheme.

Here we observe that since their first subprotocol is a statistical WI argument of knowledge given by the verifier, it can be instantiated under general complexity-theoretic assumptions only requiring a first round from prover to verifier. Indeed this message is needed to establish the parameters for a statistically hiding commitment scheme to be used in the statistical WI argument. Therefore, the resulting construction can be round optimal only when using number-theoretic assumptions, that can be used to implement the statistical WI proof in 3 rounds [96, 26].

Our technique based on temporary keys and simulation in two modes can potentially be applied when using FSZK differently, so that concurrent soundness could be preserved. This could be possible when FSZK is played independently of the public keys of the verifier, therefore including some session keys (which would have a role similar to the temporary keys of our technique). Then our new simulation technique could be used to maintain a main thread working in two modes (in one mode using the extracted permanent keys, in the other mode using the simulator of FSZK that use the extracted session keys).

The Case of Π_D [30]. A 4-round concurrently sound $\mathsf{c}\mathcal{ZK}$ argument Π_D in the BPK model under the existence of one-way functions only is showed in [30]. In the first round, the verifier sends a message m_v. Then in the second round the prover sends a statistically binding commitments of potential signatures of a pair of messages (under the public-key of the verifier) and a message m_p. In the third round the verifier sends a signature of $(m_v|m_p)$ (instead of the usual proof of knowledge of a secret). In the last 3 rounds $\mathcal{P}$ proves that $x \in L$ or the commitment sent in the second round corresponds to messages

$(m'|m'_0)$ and $(m'|m'_1)$ and their signatures, where $m'_0 \neq m'_1$.

Because the concurrent adversarial prover can not rewind the verifier, the above technique with signature, is sufficient to prove concurrent soundness. Indeed, signatures received in concurrent proofs always correspond to messages with a different prefix selected by the verifier. The $\mathsf{c}\mathcal{ZK}$ property of the protocol however is problematic again for the very same reasons discussed above. Indeed, the simulator does not have any signature at all when it plays the second round, and thus later on, in order to be able to complete proofs it will have to start new phases where knowledge of the signatures accumulated during previous executions is sufficient to run in straight-line. Indeed, the simulator presented in [30] rewinds the verifier when it is stuck, and produces a new transcript committing to the extracted signatures. As already explained, this makes its output distinguishable w.r.t. real executions.

We finally argue that the protocol could be adjusted to then admit a simulator that keeps a main thread. In contrast to previously discussed protocols, the main advantage of Π_D is that the verifier uses its secret keys to generate signatures of messages with different formats in different sessions. This makes problematic the attack of concurrent soundness, since the execution of concurrent sessions does not provide useful messages to cheat in a specific session. Therefore one could tweak the protocol so that the simulator needs to use the obtained signatures only at the last round. In this way, the simulator could obtain through rewinds two signatures for messages with the same structure, and could use them in the main thread and in all future sessions that correspond to that verifier.

5.3 Round-Optimal $\mathsf{c}\mathcal{ZK}$ in the BPK Model

In this section we show our round-optimal concurrently sound $\mathsf{c}\mathcal{ZK}$ argument under standard complexity-theoretic assumptions.

Overview, Techniques and Proof Intuition In light of the attacks shown in the previous section, we construct a protocol that allows a simulation strategy in which the transcript generated in main thread is kept unchanged. In order to highlight the issues that rise up when dealing with concurrent adversaries and how we solve them, we describe the protocol incrementally.

The public identity of the verifier $\mathcal{V}$ corresponds to a trapdoor commitment. Specifically, an identity pk_j is a pair $(\mathsf{pk}_j, \mathsf{tcom}_j)$, where pk_j is the public key of a two-round trapdoor commitment scheme (the first round consists of the generation of the parameters and is run by $\mathcal{V}$ itself) and tcom_j is the (trapdoor) commitment of a message. The secret key sk_j is the trapdoor associated to pk_j.

The protocol starts with $\mathcal{V}$ proving the knowledge of *a* valid opening of its identity: an opening of tcom_j under public key pk_j. Such proof of knowledge is implemented with a Σ-protocol that we denote by Σ^{id_j}. An honest verifier runs Σ^{id_j} using as witness a freshly generated opening of tcom_j ($\mathcal{V}$ can open tcom_j as any value since it knows the trapdoor trapdoor associated to pk_j). We use a trapdoor commitment scheme that has a

super-polynomial sized message space, hence, the number of possible witnesses is super polynomial as well. As we shall see later, this technique allows to rule out malleability attacks, and thus to obtain concurrent soundness.

Simultaneously, in the second round, the prover $\mathcal{P}$ first sends the commitment $\mathsf{com}_{\mathsf{op}}$ of a random string. Then it runs a Σ-protocol to prove that, either $x \in L$ or the string committed in $\mathsf{com}_{\mathsf{op}}$ is a valid opening of tcom_j. We denote such protocol as Σ^{L_j}. Requiring that $\mathcal{P}$ commits to the opening already in the second round is crucial to derive a contradiction in the proof of concurrent soundness.

The simulator can provide and accepting proof without knowing the witness for theorem "$x \in L$" as follows. It extracts the opening of tcom_j, exploiting the proof of knowledge property of the Σ-protocol Σ^{id_j}, and commits to such opening in $\mathsf{com}_{\mathsf{op}}$. The opening of tcom_j is the *permanent* secret associated to the verifier $\mathcal{V}$ playing with identity pk_j. Thus, once extracted, it allows the simulator to complete in straight-line all the proofs that are played with verifier pk_j.

However, this protocol clearly suffers of the same attack of previous works. Indeed, when the simulator plays with verifier pk_j for the first time, it does not know any opening. It can extract the opening only *after* having sent the commitment $\mathsf{com}_{\mathsf{op}}$. Such commitment will be *bad* since it will not be a commitment of a valid opening of tcom_j. Therefore, the simulator has to rewind the verifier and change the commitment $\mathsf{com}_{\mathsf{op}}$, hence changing the main thread. We overcome this problem with the following technique.

First, we require that upon each new session, $\mathcal{V}$ *freshly* generates an additional pair of parameters (k_0, k_1) for a two-round trapdoor commitment scheme, and it keeps the trapdoors (t_0, t_1) secret. Such parameters can be seen as *temporary* keys. $\mathcal{V}$ sends the pair (k_0, k_1) to $\mathcal{P}$ and proves knowledge of one of the trapdoors running an additional Σ-protocol that we denote by Σ^{trap}.

Second, the temporary keys (k_0, k_1) are used by $\mathcal{P}$ to commit to the first message of protocol Σ^{L_j}, that we denote by $\mathsf{pok}_1^{\mathsf{L}_j}$. Specifically, $\mathcal{P}$ computes two shares of $\mathsf{pok}_1^{\mathsf{L}_j}$ and commits to each share using one of the temporary keys. Both commitments will be opened only in the third round of Σ^{L_j}, precisely only after $\mathcal{P}$ observes the challenge sent by $\mathcal{V}$.

Allowing the prover to open the commitment after having seen the challenge does not harm soundness, while will give the simulator an additional way of cheating.

Concerning soundness, due to the binding of the commitment scheme, $\mathcal{P}$ is not able to take advantage of the knowledge of the challenge. Furthermore, since the parameters of the trapdoor commitment (i.e., the temporary keys) are freshly generated by $\mathcal{V}$ upon each execution, we are able to prove that $\mathcal{P}$ can not take advantage of concurrent executions with many verifiers[3]. We will prove concurrent soundness through witness extraction. We first show that there exists an extractor that obtains the witness from any accepting proof. Then we prove that the witness extracted is indeed a witness for theorem "$x \in L$",

[3]If instead parameters of the trapdoor commitments were fixed for all executions, then in the proof of soundness one can not derive a contradiction in case $\mathcal{P}$ equivocates the commitment associated to the same trapdoor used as witness in Σ^{trap} in concurrent executions.

and is not a valid opening of tcom_j.

Concerning the benefit for the simulation, Sim can cheat in the following additional way. It can play the second round (that is, the commitment $\mathsf{com}_{\mathsf{op}}$ and the trapdoor commitment of the message $\mathsf{pok}_1^{\mathsf{L}_j}$) without knowledge of neither a trapdoor nor a valid opening. Then, it extracts the trapdoor of one of the temporary keys and equivocates the commitment of Σ^{L_j} adaptively on the challenge received from the verifier. Here Sim is using the *special* honest-verifier zero knowledge property of protocol Σ^{L_j}. The key point is that Sim can compute the second round without knowledge of any witness or trapdoor and still be able to cheat in the last round after one of the trapdoors is extracted. One subtlety is that the simulator does not know which trapdoor will be extracted. Therefore, here we have to use a trapdoor commitment scheme in which any message computed with the honest procedure can be equivocated.

Clearly, we do not want that the simulator cheats always in this way. The temporary keys are freshly generated upon each session, thus rewinding $\mathcal{V}^*$ upon each session to extract such keys, causes the blow up of Sim's running time.

Therefore Sim operates in two modes. When simulating a session with a verifier whose *permanent* secret (the opening of tcom_j) is known, Sim runs in straight-line, using such secret as witness. When simulating a session with a new verifier, Sim launches a rewinding thread to extract both the permanent secret and the temporary trapdoor. The permanent secret is stored and it will be used to solve *future* sessions with the same identity. The temporary trapdoor is used to solve the *current* session, by equivocating the commitment of $\mathsf{pok}_1^{\mathsf{L}_j}$. We stress that with this technique, the simulator never changes messages previously played in the main thread.

Furthermore, to argue that the expected running time of Sim is polynomial, it is important that the view of the verifier in the two modes is identical. Therefore, the commitments sent by $\mathcal{P}$ are perfectly-hiding and the perfectly-trapdoor, and Σ^{L_j} is a perfect Σ-protocol.

The Protocol. Here we formally describe our round-optimal concurrently sound $\mathsf{c}\mathcal{ZK}$ protocol that we denote by $\Pi_{\mathsf{c}\mathcal{ZK}}$.

The public file. Let $\mathsf{TCom} = (\mathsf{TGen}, \mathsf{TSen}, \mathsf{TVer}, \mathsf{TFakeDec})$ be a two-round trapdoor commitment scheme. $\mathcal{V}$ first runs $(\mathsf{pk}_j, \mathsf{sk}) \leftarrow \mathsf{TGen}(1^n)$. Then it runs $(\mathsf{tcom}_j, z) \leftarrow \mathsf{TSen}(\mathsf{pk}_j, \tilde{m})$, for a random message $\tilde{m}$. Finally $\mathcal{V}$ publishes $j = (\mathsf{pk}_j, \mathsf{tcom}_j)$ as public identity and stores (sk, z) as secret key.

Sub-Protocols. Let $\mathsf{PHTCom} = (\mathsf{PHTGen}, \mathsf{PHTSen}, \mathsf{PHTRec}, \mathsf{PHTFakeDec})$ be a two-round perfectly-hiding trapdoor commitment scheme and $\mathsf{PHCom} = (\mathsf{PHGen}, \mathsf{PHSen}, \mathsf{PHRec})$ a two-round perfectly-hiding commitment scheme. For simplicity we assume that the public parameter pk for the scheme PHCom (the first round of the commitment scheme) is included in the identity of $\mathcal{V}$.

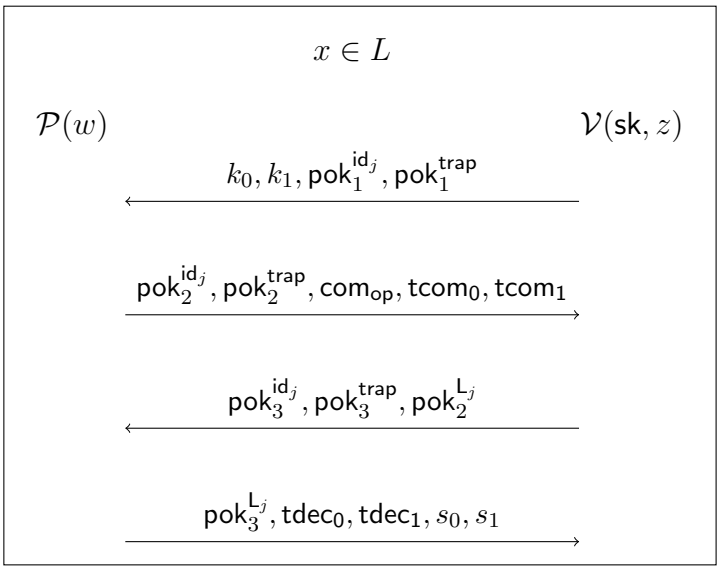

Figure 5.3: *Protocol* $\Pi_{\mathsf{c}\mathcal{ZK}}$.

Auxiliary languages. We use the following **NP** relations and in turn the respective **NP**-languages $L_{\mathsf{id}_j}, L_{\mathsf{trap}}, L_{\mathsf{op}_j}, L_j$:

- knowledge of an opening of tcom_j under parameter pk_j:
  $\mathcal{R}_{L_{\mathsf{id}_j}} = \{((\mathsf{pk}_j, \mathsf{tcom}_j), (\mathtt{open}, m)) : \mathsf{TVer}(\mathsf{pk}_j, \mathsf{tcom}_j, \mathtt{open}, m) = 1\}$;
- knowledge of the trapdoor t associated to one of the temporary parameters (k_0, k_1):
  $\mathcal{R}_{L_{\mathsf{trap}}} = \{((k_0, k_1), t) : (k_0, t) \leftarrow \mathsf{PHTGen}(1^n) \vee (k_1, t) \leftarrow \mathsf{PHTGen}(1^n)\}$;
- knowledge of the string v committed in $\mathsf{com}_{\mathsf{op}}$ which is a valid opening of tcom_j (v can be seen as the concatenation $\mathsf{dec}'||\mathsf{m}'$ where dec' is the opening information for tcom_j and m' is the message):
  $\mathcal{R}_{L_{\mathsf{op}_j}} = \{((\mathsf{com}_{\mathsf{op}}, \mathsf{pk}_j, \mathsf{tcom}_j), (v, d)) : \mathsf{PHRec}(pk, \mathsf{com}_{\mathsf{op}}, d, v = (\mathsf{dec}'||\mathsf{m}')) = 1 \wedge$
  $\mathsf{TVer}(\mathsf{pk}_j, \mathsf{tcom}_j, \mathsf{dec}', \mathsf{m}') = 1\}$;
- knowledge of the witness for $x \in L$, or knowledge of the witness for $(\mathsf{com}_{\mathsf{op}}, \mathsf{pk}_j, \mathsf{tcom}_j) \in L_{\mathsf{op}_j}$:
  $\mathcal{R}_{\mathsf{L}_j} := \mathcal{R}_L \vee \mathcal{R}_{L_{\mathsf{op}_j}} = \{X = (x, \mathsf{com}_{\mathsf{op}}, \mathsf{pk}_j, \mathsf{tcom}_j),\ W = (y, v, d)) : (x, w) \in R_L \vee ((\mathsf{com}_{\mathsf{op}}, \mathsf{pk}_j, \mathsf{tcom}_j), (v, d)) \in \mathcal{R}_{L_{\mathsf{op}_j}}\}$.

Σ-Protocols. We will use Σ-protocols for all languages showed above. We denote by $\Sigma^{\mathsf{id}_j} = (\mathsf{pok}_1^{\mathsf{id}_j}, \mathsf{pok}_2^{\mathsf{id}_j}, \mathsf{pok}_3^{\mathsf{id}_j})$ the Σ-protocol run by $\mathcal{V}$ to prove knowledge of an opening of its identity $\mathsf{pk}_j = (\mathsf{pk}_j, \mathsf{tcom}_j)$ (relation $\mathcal{R}_{L_{\mathsf{id}_j}}$) and by $\Sigma^{\mathsf{trap}} = (\mathsf{pok}_1^{\mathsf{trap}}, \mathsf{pok}_2^{\mathsf{trap}}, \mathsf{pok}_3^{\mathsf{trap}})$ the Σ-protocol run by $\mathcal{V}$ to prove knowledge of a temporary trapdoor (relation $\mathcal{R}_{L_{\mathsf{trap}}}$).

We denote by $\Sigma^{\mathsf{L}_j} = (\mathsf{pok}_1^{\mathsf{L}_j}, \mathsf{pok}_2^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j})$ the perfect Σ-protocol run by $\mathcal{P}$ for instances of $\mathcal{R}_{\mathsf{L}_j}$ when interacting with the verifier with identity pk_j.

The protocol is depicted in Figure 5.4. A graphic representation is given in Figure 5.3. By noticing that Blum's protocol [12] is a perfect Σ-protocol (when the first message is computed with a perfectly-hiding commitment scheme) for **NP** languages, we conclude that Protocol $\Pi_{\mathsf{c}\mathcal{Z}\mathcal{K}}$ is a black-box perfect $\mathsf{c}\mathcal{Z}\mathcal{K}$ for all **NP**.

Theorem 12. *If $\Sigma^{\mathsf{L_j}}$ is a perfect Σ-protocol, if $\Sigma^{\mathsf{id}_j}, \Sigma^{\mathsf{trap}}$ are Σ-protocols, if* PHCom *is a two-round perfectly-hiding commitment scheme,* PHTCom *is a two-round perfectly-hiding trapdoor commitment scheme and* TCom *is a two-round trapdoor commitment scheme, then protocol $\Pi_{\mathsf{c}\mathcal{Z}\mathcal{K}}$ is a 4-round concurrently sound black-box perfect $\mathsf{c}\mathcal{Z}\mathcal{K}$ argument in the* BPK *model for all* **NP**.

5.3.1 Security Proof

Concurrent Zero Knowledge of $\Pi_{\mathsf{c}\mathcal{Z}\mathcal{K}}$

We show an expected PPT simulator Sim that with oracle access to $\mathcal{V}^*$ outputs a transcript that is indistinguishable from the output of the execution between $\mathcal{P}$ and $\mathcal{V}^*$. In the following we first describe the simulator Sim and we prove that its expected running time is polynomial. Then we prove the indistinguishability of the view generated by Sim

The Simulator. The simulator Sim is depicted in Figure 5.5. Sim gets as input the vector of theorems $(x_1, \ldots, x_{\mathtt{poly}(n)})$ and the vector of identities $((\mathsf{pk}_1, \mathsf{tcom}_1)$,..., $(\mathsf{pk}_{\mathtt{poly}(n)}, \mathsf{tcom}_{\mathtt{poly}(n)}))$.

Sim maintains three global variables $h, \mathcal{T}, R_{\mathcal{V}^*}$. Variable h maintains the transcript of the Main Thread, variable $\mathcal{T}$ maintains the permanent secrets extracted so far: the j-th entry of $\mathcal{T}$ is the permanent secret – denoted by $\mathtt{open}_j$ – of the j-th identity $(\mathsf{pk}_j, \mathsf{tcom}_j)$. $R_{\mathcal{V}^*}$ is the randomness with which $\mathcal{V}^*$ is activated.

Sim has oracle access to $\mathcal{V}^*$ and receives queries in the form $(i, j, m_{\mathcal{V}})$. Here i is the index of the theorem to be proved, j is the index of the identity of $\mathcal{V}^*$ which corresponds to the pair $(\mathsf{pk}_j, \mathsf{tcom}_j)$, $m_{\mathcal{V}}$ is $\mathcal{V}^*$'s message. Value $m_{\mathcal{V}}$ is either the first round, which we denote by $V1 = (k_0, k_1, \mathsf{pok}_1^{\mathsf{id}_j}, \mathsf{pok}_1^{\mathsf{trap}})$, or the third round, which we denote by $V3 = (\mathsf{pok}_3^{\mathsf{id}_j}, \mathsf{pok}_3^{\mathsf{trap}}, \mathsf{pok}_2^{\mathsf{L}_j})$. $\mathcal{V}^*$ can open concurrently many sessions with the same identity. For convenience, we identify each session with a session id sid.

When $\mathcal{V}^*$ initiates a *new* session, it sends a query $(\mathsf{sid}, i, j, V1)$ to Sim. Sim checks if identity j has been already solved, namely, if there exists $\mathtt{open}_j \in \mathcal{T}$. If this is the case, then Sim computes $\mathsf{com}_{\mathsf{op}} = \mathsf{PHSen}(pk, \mathtt{open}_j)$, and later honestly completes the sigma protocol Σ^{L_j} using $\mathtt{open}_j$ as witness. In this case, we say that session sid is solved.

If identity j is unsolved, Sim computes $\mathsf{com}_{\mathsf{op}} = \mathsf{PHSen}(pk, r)$ for a random string r, and commit to random shares in $\mathsf{tcom}_0, \mathsf{tcom}_1$. In this case we say that session sid is not solved.

When $\mathcal{V}^*$ sends a third round $(\mathsf{sid}, i, j, V3)$ and session sid is not solved Sim must extract the temporary trapdoor for session sid, in order to equivocate one of the commit-

Common input: the public file F, an n-bit string $x \in L$ and index j specifying the j-th entry of F, i.e. $\mathsf{pk}_j = (\mathsf{pk}_j, \mathsf{tcom}_j) for the trapdoor commitment and pk$ per the perfectly hiding commitment.
$\mathcal{P}$'s private input: a witness w for $x \in L$.
$\mathcal{V}$'s private input: the trapdoor key sk and z which is the auxiliary information needed for equivocation.
$\mathcal{V}$-round-1:
- $(k_0, t_0) \xleftarrow{\$} \mathsf{PHTRec}(1^n)$; $(k_1, t_1) \xleftarrow{\$} \mathsf{PHTRec}(1^n)$;
- compute $\mathsf{pok}_1^{\mathsf{id}_j}$ and $\mathsf{pok}_1^{\mathsf{trap}}$;
- send k_0, k_1, $\mathsf{pok}_1^{\mathsf{id}_j}$, $\mathsf{pok}_1^{\mathsf{trap}}$ to $\mathcal{P}$.

$\mathcal{P}$-round-2:
- $(\mathsf{com}_{\mathsf{op}}, d) \leftarrow \mathsf{PHSen}(pk, v)$ for a randomly chosen string v; compute $\mathsf{pok}_2^{\mathsf{id}_j}$, $\mathsf{pok}_2^{\mathsf{trap}}$;
- compute $\mathsf{pok}_1^{\mathsf{L}_j}$ and shares s_0, s_1 such that $s_0 \oplus s_1 = \mathsf{pok}_1^{\mathsf{L}_j}$;
- $(\mathsf{tcom}_0, \mathsf{tdec}_0) \leftarrow \mathsf{PHTSen}(k_0, s_0)$, $(\mathsf{tcom}_1, \mathsf{tdec}_1) \leftarrow \mathsf{PHTSen}(k_1, s_1)$;
- send $\mathsf{pok}_2^{\mathsf{id}_j}$, $\mathsf{pok}_2^{\mathsf{trap}}$, $\mathsf{com}_{\mathsf{op}}$, $\mathsf{tcom}_0, \mathsf{tcom}_1$ to $\mathcal{V}$.

$\mathcal{V}$-round-3:
- pick a random n-bit string m';
- compute $(m' || wdec) \leftarrow \mathsf{TFakeDec}(\mathsf{sk}, z, m')$;
- compute $\mathsf{pok}_3^{\mathsf{id}_j}$ using as witness $(m', wdec)$;
- compute $\mathsf{pok}_3^{\mathsf{trap}}$ using as witness t_e for a randomly selected bit e;
- compute $\mathsf{pok}_2^{\mathsf{L}_j}$;
- send $\mathsf{pok}_3^{\mathsf{id}_j}, \mathsf{pok}_3^{\mathsf{trap}}, \mathsf{pok}_2^{\mathsf{L}_j}$ to $\mathcal{P}$.

$\mathcal{P}$-round-4:
- verify that $(\mathsf{pok}_1^{\mathsf{id}_j}, \mathsf{pok}_2^{\mathsf{id}_j}, \mathsf{pok}_3^{\mathsf{id}_j})$ is an accepting transcript of Σ^{id_j} for the statement $(\mathsf{pk}_j, \mathsf{tcom}_j) \in L_{\mathsf{id}_j}$, if not abort;
- verify that $(\mathsf{pok}_1^{\mathsf{trap}}, \mathsf{pok}_2^{\mathsf{trap}}, \mathsf{pok}_3^{\mathsf{trap}})$ is an accepting transcript of Σ^{trap} for the statement $(k_0, k_1) \in L_{\mathsf{trap}}$, if not abort;
- compute $\mathsf{pok}_3^{\mathsf{L}_j}$ using the witness w;
- send $\mathsf{pok}_3^{\mathsf{L}_j}, \mathsf{tdec}_0, \mathsf{tdec}_1, s_0, s_1$ to $\mathcal{V}$.

$\mathcal{V}$-decision: if $\mathsf{PHTRec}(k_0, \mathsf{tcom}_0, \mathsf{tdec}_0, s_0) = 1 \wedge \mathsf{PHTRec}(k_1, \mathsf{tcom}_1, \mathsf{tdec}_1, s_1) = 1$ then $\mathsf{pok}_1^{\mathsf{L}_j} \leftarrow s_0 \oplus s_1$ and accept iff $(\mathsf{pok}_1^{\mathsf{L}_j}, \mathsf{pok}_2^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j})$ is an accepting transcript of Σ^{L_j} for the statement $(x, \mathsf{com}_{\mathsf{op}}, \mathsf{pk}_j, \mathsf{tcom}_j) \in L_j$; else, abort.

Figure 5.4: $\Pi_{\mathsf{c}\mathcal{ZK}}$*: 4-round concurrently-sound* $\mathsf{c}\mathcal{ZK}$ *in the* BPK *model for all* **NP**.

ments $\mathsf{tcom}_0, \mathsf{tcom}_1$. Sim pauses the Main Thread and starts the extraction thread – it launches procedure Extract – which serves to extract both the temporary trapdoor (t_0 or t_1), and the opening of tcom_j ($\mathtt{open}_j$). This is done by rewinding $\mathcal{V}^*$ in the execution of Σ-protocols Σ^{id_j}, Σ^{trap} respectively (details on procedure Extract will be provided soon).

Once the extraction of the temporary trapdoor is successfully completed, Sim computes an accepting last round of $\Pi_{\mathsf{c}\mathcal{ZK}}$ (i.e., $\mathsf{pok}_3^{\mathsf{L}_j}, \mathsf{tdec}_0, \mathsf{tdec}_1, s_0, s_1$) as follows. It computes $\mathsf{pok}_1^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j}$ by running the *special* honest-verifier zero knowledge simulator hvSim of Σ^{L_j}. Then it uses the extracted trapdoor to equivocate one of the commitments so that the xor of the openings $s_0 \oplus s_1$ gives the message $\mathsf{pok}_1^{\mathsf{L}_j}$ calculated as above.

Procedure Extract focuses on extracting one of the temporary trapdoors and the opening of tcom_j for session sid only. It rewinds $\mathcal{V}^*$ up to the second round of the session sid, and sends fresh values $\widetilde{\mathsf{pok}}_2^{\mathsf{id}_j}, \widetilde{\mathsf{pok}}_2^{\mathsf{trap}}$ (the challenges of protocols Σ^{id_j}, Σ^{trap}) to $\mathcal{V}^*$. The goal is to simulate the view of $\mathcal{V}^*$ till when she answers with a third round for session sid, so that the witnesses of both Σ-protocols are extracted.

To simulate such view, procedure Extract has to simulate the execution of all sessions played by $\mathcal{V}^*$ between the second and the third round of session sid. Such sessions are simulated in straight-line using the permanent keys extracted so far. Note that, there is a difference between the way such sessions are handled in the Main Thread, versus the rewinding thread. In the Main Thread, the sessions appearing between the second and the third round of session sid, might have been solved using temporary keys. Whereas, in the rewinding threads all such sessions are solved using the permanent key. As we shall see later, in order to argue about the running time of Sim we need that the view of $\mathcal{V}^*$ in the above two modes is identically distributed. This allows us to argue that if in the Main Thread a session is completed with non-negligible probability, so it will be in each rewinding thread. Here we are crucially helped by the fact that the protocols played by $\mathcal{P}$ are perfectly-hiding, perfectly-trapdoor and perfectly-WI.

Observe that, in the Main Thread, $\mathcal{V}^*$ did not complete any *unsolved* session, say sid$'$, before sending the third message of session sid (this is obviously true, since otherwise Sim would have called Extract on session sid$'$ instead). However, in each rewinding attempt performed by Extract, $\mathcal{V}^*$ receives new values $\widetilde{\mathsf{pok}}_2^{\mathsf{id}_j}, \widetilde{\mathsf{pok}}_2^{\mathsf{trap}}$, thus $\mathcal{V}^*$ could complete *unsolved* sessions that were not completed in the Main Thread, before sending the third round of sid.

In such case, the procedure Extract tries to solve the unsolved session in order to obtain the third round of the target unsolved session. The [19] guarantees that the running time required for such session is still polynomial.

Procedure Extract is shown in Figure 5.6.

Claim 5.3.1. Sim *runs in expected polynomial time.*

Proof. Due to the temporary trapdoors, the number of extractions (invocations of procedure Extract) is not upper-bounded by the number of identities, but it can match the number of sessions that $\mathcal{V}^*$ opens in the Main Thread. In the following we formally show that the expected running time of Sim is polynomial.

When $\mathcal{V}^*$ opens a new session by querying Sim with $(\mathsf{sid}, i, j, V1)$, there are two possible cases:

Case 1. Identity j is solved. If the identity has been solved, Sim computes the second round as the honest prover, except that it computes $\mathsf{com}_{\mathsf{op}}$ as the commitment of open_j.

Then, when $\mathcal{V}^*$ sends the third round $V3$, Sim proceeds in straight-line using the witness $\mathtt{open}_j$. Therefore, for solved identities the running time of Sim is the same as the running time of $\mathcal{P}$. We denote the running time needed for solved sessions by $\mathsf{t}_{\mathsf{solved}} = \mathtt{poly}(n)$.

Case 2. Identity j that is not solved yet. If the identity has not been solved, Sim computes the second round committing to random strings. We denote the running time necessary to compute this first round by $\mathsf{t}_{P_{\mathsf{rszk}}} = \mathtt{poly}(n)$.

Then, when $\mathcal{V}^*$ correctly sends the message $V3$, Sim activates procedure $\mathsf{Extract}$ to extract the temporary trapdoor and the permanent secret. Procedure $\mathsf{Extract}$ takes as input the transcript of the main thread obtained till the second round of session sid.

Procedure $\mathsf{Extract}$ focuses on session sid only. Its goal is to extract the witness used in protocols $\Sigma^{\mathsf{id}_j}, \Sigma^{\mathsf{trap}}$ run in session sid. Towards this goal, it changes the challenges for protocols $\Sigma^{\mathsf{id}_j}, \Sigma^{\mathsf{trap}}$ sent in the second round of session sid, hoping to reach another accepting third round, thus extracting both witnesses. However, in between the second and the third round of session sid, the malicious verifier $\mathcal{V}^*$ might have opened other sessions that must be simulated as well. Such sessions are simulated as in procedure Main Thread with the following exception. While in the main thread they were solved using the temporary trapdoor, in the rewinding thread are solved in straight-line using the permanent secret. When $\mathcal{V}^*$ sends the third round of a session that is *unsolved*, the procedure $\mathsf{Extract}$ tries to solve such session with the goal of reaching the third round of session sid (and on the way it collects new permanent trapdoors which allows straight-line simulation). When $\mathcal{V}^*$ sends a valid third round for session sid, procedure $\mathsf{Extract}$ extracts the witnesses used in Σ^{id_j} (an opening of tcom_j) and Σ^{trap} (the temporary trapdoor). It stores the opening in the table $\mathcal{T}$ and returns the trapdoor to the Main Thread. Procedure $\mathsf{Extract}$ does not call itself recursively, thus we only have to bound the number of rewinding attempts needed to extract the witnesses.

We now argue that the expected number of rewinding attempts needed to extract the trapdoor (and thus the opening) is polynomial. The crucial point that helps our analysis is that the view of $\mathcal{V}^*$ in the main thread is identical to its view in each rewinding attempt. This is due to the fact that commitment functions $\mathsf{PHCom}, \mathsf{PHTCom}$ are perfectly-hiding, PHTCom is perfectly-trapdoor, and the Σ-protocol Σ^{L_j} is perfect.

Therefore, if $\mathcal{V}^*$ has reached the third round of session sid in the main thread, without terminating any other unsolved session, with non-negligible probability, then in each rewinding thread, $\mathcal{V}^*$ will do the same, with the same probability. Hence, the expected number of rewinding attempts required to reach a valid third step for sid, is polynomial.

More formally, let s be an upper-bound the number of sessions opened by $\mathcal{V}^*$. Let $\zeta^{\mathsf{sid}} = \zeta(R_{\mathcal{V}^*}, \mathsf{sid})$ be the probability over the coin tosses by Sim that $\mathcal{V}^*$, activated with randomness $R_{\mathcal{V}^*}$, correctly plays messages $\mathsf{pok}_3^{\mathsf{id}_j}$ and $\mathsf{pok}_3^{\mathsf{trap}}$ for the third round of the unsolved session sid.

When Sim reaches the third round of such session, it activates procedure $\mathsf{Extract}$. All the work up to this point concerns either sessions activated with identities already solved (dealing with these sessions takes time $\mathsf{t}_{\mathsf{solved}}$) or sessions for which only the first prover's message has been played (dealing with these sessions takes time $\mathsf{t}_{P_{\mathsf{rszk}}}$). This

amount of work is upper bounded by $(s \cdot \mathsf{t}_{P_{\mathsf{rszk}}} + s \cdot \mathsf{t}_{\mathsf{solved}})$ and is repeated by procedure Extract upon each rewinding attempt. Procedure Extract generates a transcript that is identical to the one generated by Sim in the main thread.

Hence the expected running time of one invocation of the procedure Extract to solve a session sid is bounded by:

$$\mathsf{t}_{\mathsf{solvesid}} \leq \zeta^{\mathsf{sid}} \left[\frac{s}{\zeta^{\mathsf{sid}}} \cdot (\mathsf{t}_{P_{\mathsf{rszk}}} + \mathsf{t}_{\mathsf{solved}}) \right] = \mathtt{poly}(n)$$

Thus, the total expected running time of Sim is upper bounded by:

$$\mathsf{t}_{\mathsf{Sim}} \leq \sum_{\mathsf{sid}=1}^{s} (\mathsf{t}_{P_{\mathsf{rszk}}} + \mathsf{t}_{\mathsf{solvesid}}) + \mathtt{poly}(n) \cdot \mathsf{t}_{\mathsf{solved}} = \mathtt{poly}(n)$$

□

Claim 5.3.2. *The simulator* Sim *outputs* ABORT *with negligible probability.*

Proof. Sim aborts only when procedure Extract fails in returning the trapdoor key to solve a session sid. Procedure Extract fails in outputting the witnesses in two cases:

- None of the 2^n attempts is successful. This case happens with negligible probability.
- $\mathcal{V}^*$ violates the *special* soundness of protocols $\Sigma^{\mathsf{id}_j}, \Sigma^{\mathsf{trap}}$. Since the latter are Σ-protocols, this event happens with negligible probability only.

□

Claim 5.3.3. *The view generated by* Sim *having oracle access to* $\mathcal{V}^*$ *and input the set* $\bar{x} = \{x_1, \ldots, x_n\}$ *is perfectly-indistinguishable from the view generated by the interaction between* $\mathcal{V}^*$ *and* $\mathcal{P}$.

Proof. The proof goes by hybrid arguments.

H_0: In this hybrid Sim has in input the vector of witnesses $\bar{w} = w_1, \ldots, w_n$ and follows the procedure of the honest $\mathcal{P}$. The output of this experiment is distributed identically to the output of the real game played between $\mathcal{P}$ and $\mathcal{V}^*$.

H_1: This hybrid is the same as H_0 except that here Sim runs the procedure Extract. That is, when $\mathcal{V}^*$ reaches the third round in a session sid that is not solved yet, Sim activates the procedure Extract. However, all the prover's messages are computed according to the honest prover procedure and the openings extracted are never used. The only difference between hybrid H_0 and H_1 is that H_1 runs in expected polynomial time and that Sim in H_1 can abort with negligible probability (due to the possible failure of procedure Extract). The two experiments are statistically close. Note that, although the output of the experiments is statistically close, the view generated by Sim (when it does not abort) in H_1, is identical to the view generated by Sim in H_0.

H_2**:** This hybrid is the same as H_1 except that in H_2, when $\mathcal{V}^*$ starts a new session sid with identity j (i.e. $\mathcal{V}^*$ sends the message of the first round $V1$) that is already solved (i.e., there exists $\mathtt{open}_j \in \mathcal{T}$) Sim computes $\mathsf{com}_{\mathsf{op}}$ as the commitment of the opening $\mathtt{open}_j$ instead of a random string. However, as for the other sessions, Sim still computes the last message (the third message of protocol Σ^{L_j}) of session sid using the witness for language L, as the honest prover. The only difference between experiment H_2 and H_1 is in the string committed in $\mathsf{com}_{\mathsf{op}}$. By the perfectly-hiding property of commitment scheme PHCom hybrids H_1 and H_2 are identical.

H_3**:** This hybrid is the same as H_2 except that here for a new session sid in which $\mathcal{V}^*$ plays with an identity j that is already solved, Sim computes the third round of the protocol Σ^{L_j} by using as witness the decommitment of $\mathsf{com}_{\mathsf{op}}$ (which consists of the opening $\mathtt{open}_j$ of tcom_j). The only difference between hybrids H_2 and H_3 is in the witness used to compute the last message of Σ^{L_j}. Since Σ^{L_j} is a perfect Σ-protocol, and thus perfect $\mathcal{WI}$, hybrids H_2 and H_3 are identical.

H_4**:** This hybrid is the same as H_3 except that here when $\mathcal{V}^*$ plays the third round of a session sid that is unsolved, Sim uses the temporary trapdoor extracted by $\mathsf{Extract}$ to compute the last message. More specifically, Sim computes all the messages accordingly to H_3 but it opens commitment $\mathsf{tcom}_{\mathsf{e}}$ as $\mathsf{tdec}_{\mathsf{e}}$ using algorithm $\mathsf{PHTFakeDec}$ and the trapdoor key t_e instead of algorithm PHTSen. Note that Sim here still computes the message $\mathsf{pok}_3^{\mathsf{L}_j}$ using the witness w. The only difference between hybrid H_3 and H_4 is that in hybrid H_4 one decommitment is computed using the trapdoor, instead of the honest decommitment procedure. Due to the perfect trapdoorness of PHTCom, hybrids H_3 and H_4 are identical.

H_5**:** This hybrid is the same as H_4 except that here when $\mathcal{V}^*$ plays the third round of a session sid that is unsolved, Sim computes the last prover message using the *special* honest-verifier zero-knowledge simulator of Σ^{L_j}. Specifically, Sim computes the message $(\mathsf{pok}_1^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j}) \leftarrow \mathsf{hvSim}(\mathsf{pok}_2^{\mathsf{L}_j})$. Here Sim is not using any witness to prove the theorem. Since Σ^{L_j} is a perfect Σ-protocol, the honest-verifier zero-knowledge simulator hvSim generates a transcript that is identical to the one generated by the honest prover. Therefore hybrids H_4 and H_5 are identical.

By noticing that this is the description of the simulator Sim, the proof of the claim is completed.

□

Concurrent Soundness of $\Pi_{\mathsf{c}\mathcal{ZK}}$

In order to prove concurrent soundness we show that if a malicious prover $\mathcal{P}^*$ provides an accepting transcript then we can efficiently extract the witness for the theorem "$x \in L$", therefore contradicting the fact that "$x \notin L$".

The proof is divided in two parts.

First, we prove that if a malicious prover $\mathcal{P}^*$ provides an accepting transcript, the extractor E obtains the witness with high probability in expected polynomial time. This proof does not follow straightforwardly from the proof of knowledge property of the Σ-protocol Σ^{L_j}. The reason is our novel use of temporary trapdoors. Recall that according to $\Pi_{\mathsf{c}\mathcal{ZK}}$, $\mathcal{P}^*$ actually delivers the first message of Σ^{L_j} ($\mathsf{pok}_1^{\mathsf{L}_j}$) only in the last round (by opening commitments $\mathsf{tcom}_0, \mathsf{tcom}_1$). Namely, only after it sees the message $\mathsf{pok}_2^{\mathsf{L}_j}$ sent by $\mathcal{V}$. This is the crucial idea that allows the simulator to cheat in the last round without changing the transcript played so far. However, when proving soundness we have to consider a malicious prover that changes the transcript of Σ^{L_j} adaptively on the challenge $\mathsf{pok}_2^{\mathsf{L}_j}$ as well. In this case the extraction of the witness fails (recall, to extract a witness two distinct transcripts that have the *same* first message $\mathsf{pok}_1^{\mathsf{L}_j}$ are required). We will prove that, due to the binding of the trapdoor commitment PHTCom used by $\mathcal{P}^*$ to compute $\mathsf{tcom}_0, \mathsf{tcom}_1$, and to the witness indistinguishability of protocol Σ^{trap} (where $\mathcal{V}$ proves knowledge of one of the temporary trapdoors), this event happens with negligible probability only. In the reduction to the witness indistinguishability property of Σ^{trap}, we do not have to deal with malleability attacks. In a malleability attack $\mathcal{P}^*$ exploits the concurrent executions with the same verifier $\mathcal{V}$ (by same verifier we mean a verifier with the same identity) to compute the messages of the accepting session. The key point here is that the temporary keys used to run protocol Σ^{trap} are independently and randomly chosen upon each new session. Thus, for the proof of soundness it is crucial that the keys for the trapdoor commitment are *temporary*, instead of being part of the public file. The first part of the security proof is given in Claim 5.3.4 and Claim 5.3.5.

Second, we prove that the witness extracted by E is indeed a witness for the theorem $x \in L$. Recall that in protocol $\Pi_{\mathsf{c}\mathcal{ZK}}$, the prover $\mathcal{P}^*$ provides a proof of knowledge for the language L_j. The first part of the security proof guarantees that E obtains the witness W for the language L_j with overwhelming probability. However, since $L_j = L \vee L_{\mathsf{op}_j}$, the extracted witness W can correspond either to the witness w such that $(x, w) \in \mathcal{R}_L$ or to the openings (v, d) such that $((\mathsf{com}_{\mathsf{op}}, \mathsf{pk}_j, \mathsf{tcom}_j), (v, d)) \in \mathcal{R}_{L_{\mathsf{op}_j}}$.

We prove that, due to trapdoorness and the binding property of TCom, and due to the witness indistinguishability property of the Σ-protocol Σ^{id_j} run by $\mathcal{V}$, the probability that the extracted witness is a valid opening of tcom_j is negligible. The most delicate point of this second part is taking care of the malleability attack that $\mathcal{P}^*$ can mount exploiting concurrent executions of Σ^{id_j} with $\mathcal{V}$. Indeed, since the identity is fixed in the public file, in the concurrent executions $\mathcal{V}$ proves always the same theorem, that is, knowledge of the opening of the same commitment tcom_j. We are able to rule out malleability attacks using the following facts:

1. $\mathcal{P}^*$ sends $\mathsf{com}_{\mathsf{op}}$ that is the commitment to the (possible) opening of tcom_j, already in the first message of Σ^{L_j}. Therefore, $\mathcal{P}^*$ can not maul executions of Σ^{id_j} played *concurrently* with Σ^{L_j}, unless: 1) the binding of $\mathsf{com}_{\mathsf{op}}$ is broken (this happens with negligible probability due to the binding of PHCom); 2) $\mathcal{P}^*$ guesses the witness used by $\mathcal{V}$ in a concurrent session and commits to it in $\mathsf{com}_{\mathsf{op}}$ (this happens with

negligible probability due to the next fact).

2. there exists a super-polynomial number of witnesses for L_{id_j} (the space of possible openings of tcom_j), that can be used by $\mathcal{V}$ to run Σ^{id_j}. Thus, $\mathcal{P}^*$ cannot predict which witness will be used in the concurrent execution.

The second part is formally proved in Claim 5.3.6. In the following, we first show the extractor E and then we prove that E satisfies Claims 5.3.4, 5.3.5 and 5.3.6.

The Extractor The extractor is depicted in Fig. 5.7. It runs in two phases. In the first phase, it runs identically as the honest verifier $\mathcal{V}$. For each accepting transcript observed in the first phase, it starts the extraction phase. It rewinds $\mathcal{P}^*$ and changes the challenge of Σ^{L_j} in order to get another accepting transcript, so that, due to the special soundness of Σ^{L_j}, it extracts the witness.

In the following, $\mathcal{P}^*$ is an s-concurrent malicious prover, that opens s concurrent sessions with an honest verifier $\mathcal{V}$ which plays all sessions using the identity $j = (\mathsf{pk}_j, \mathsf{tcom}_j)$ and independent random tapes.

Claim 5.3.4. *E runs in expected polynomial time in n.*

Proof. In the first phase E runs the honest verifier procedure for up to $s = \mathtt{poly}(n)$ sessions. Let us denote the running time of $\mathcal{V}$ in one session by $\mathsf{t}_{\mathcal{V}} = \mathtt{poly}(n)$. The extraction phase is executed only for sessions in which $\mathcal{P}^*$ has provided an accepting proof in the first phase, and it is done sequentially for one session at time. Let denote by $\zeta^i = \zeta(R_{\mathcal{P}^*}, i)$ the probability over the coins tossed by E, that $\mathcal{P}^*$, activated with randomness $R_{\mathcal{P}^*}$, provides an accepting transcript $(\mathsf{pok}_1^{\mathsf{L}_j}, \mathsf{pok}_2^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j})$ in session i in the first phase. At each rewind, E uses the same randomness of the first phase and only changes the message $\mathsf{pok}_2^{\mathsf{L}_j}$ with a new message $\widetilde{\mathsf{pok}}_2^{\mathsf{L}_j}$. The view of $\mathcal{P}^*$ in each rewind is distributed identically as the view provided in the first phase. Hence, upon each rewind, the probability of getting another accepting transcript is still ζ^i. Thus, obtaining another transcript takes $1/\zeta^i$ expected number of attempts. Each attempt takes time at most $\mathsf{t}_{\mathtt{rew}} = s \cdot \mathsf{t}_{\mathcal{V}} = \mathtt{poly}(n)$.

The total expected running time, consists in the running time of the first phase, that is $s \cdot \mathsf{t}_{\mathcal{V}}$, plus the time to run the extraction phase for each accepting session i:

$$\mathsf{t}_E = \mathsf{t}_{\mathsf{firstphase}} + \sum_{i=1}^{s} \mathsf{t}_{\mathsf{extraction}i} = s \cdot \mathsf{t}_{\mathcal{V}} + \sum_{i=1}^{s} \zeta^i \left[\frac{1}{\zeta^i} \cdot \mathsf{t}_{\mathtt{rew}} \right] \simeq \mathtt{poly}(n).$$

□

Claim 5.3.5. *If a session i is accepting in the first phase, then in the extraction phase, E outputs the witness W of relation $\mathcal{R}_{L_{L_j}} = \mathcal{R}_L \vee \mathcal{R}_{L_{\mathsf{op}_j}}$ for session i with overwhelming probability.*

Proof. We prove this claim by showing that, given that $\mathcal{P}^*$ provided an accepting transcript in the first phase, the probability that E aborts is negligible. Indeed procedure E either extracts the witness or it aborts.

By assumption, in the first phase $\mathcal{P}^*$ provided an accepting transcript $\{\mathsf{pok}_1^{\mathsf{L}_j} = (s_0 \oplus s_1), \mathsf{pok}_2^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j}\}$ for session i. Here, s_0, s_1 are the strings committed to in $\mathsf{tcom}_0, \mathsf{tcom}_1$, obtained from decommitments $\mathsf{tdec}_0, \mathsf{tdec}_1$ respectively.

The extractor aborts when it obtains two valid transcripts of protocol Σ^{L_j} for session i, but they do not share the same first message. This happens when $\mathcal{P}^*$ provides two distinct openings for the same trapdoor commitment. Specifically, from the first phase, E obtains the transcript $\mathsf{tcom}_0, \mathsf{tcom}_1, \mathsf{tdec}_0, \mathsf{tdec}_1, s_0, s_1, \{(\mathsf{pok}_1^{\mathsf{L}_j} = s_0 \oplus s_1), \mathsf{pok}_2^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j}\}$ where s_0, s_1 are the strings decommitted from $\mathsf{tcom}_0, \mathsf{tcom}_1$ (verified with $\mathsf{tdec}_0, \mathsf{tdec}_1$) and $s_0 \oplus s_1 = \mathsf{pok}_1^{\mathsf{L}_j}$ such that $\mathsf{pok}_1^{\mathsf{L}_j}, \mathsf{pok}_2^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j}$ is an accepting transcript for Σ^{L_j}. From the extraction phase, E obtains $\mathsf{tcom}_0, \mathsf{tcom}_1, \widetilde{\mathsf{tdec}}_0, \widetilde{\mathsf{tdec}}_1, \tilde{s}_0, \tilde{s}_1, \{\widetilde{\mathsf{pok}}_1^{\mathsf{L}_j} = (\tilde{s}_0 \oplus \tilde{s}_1), \widetilde{\mathsf{pok}}_2^{\mathsf{L}_j} \widetilde{\mathsf{pok}}_3^{\mathsf{L}_j}\}$ where $\tilde{s}_0, \tilde{s}_1$ are the strings decommitted from $\mathsf{tcom}_0, \mathsf{tcom}_1$ (verified with $\widetilde{\mathsf{tdec}}_0, \widetilde{\mathsf{tdec}}_1$) and $\tilde{s}_0 \oplus \tilde{s}_1 = \widetilde{\mathsf{pok}}_1^{\mathsf{L}_j}$ is such that $\widetilde{\mathsf{pok}}_1^{\mathsf{L}_j}, \widetilde{\mathsf{pok}}_2^{\mathsf{L}_j}, \widetilde{\mathsf{pok}}_3^{\mathsf{L}_j}$ corresponds to an accepting transcript *and* $\mathsf{pok}_1^{\mathsf{L}_j} \neq \widetilde{\mathsf{pok}}_1^{\mathsf{L}_j}$.

Assume, toward a contradiction, that this event happens with non-negligible probability p. Then we construct an adversary S^* who breaks the binding of the trapdoor commitment scheme PHTCom with probability p/s. S^* receives the parameter pk from PHTRec and works as follows. It uniformly chooses one of the sessions, let us denote it by i. It runs the first phase of the extractor, i.e., it prepares the identity and honestly simulates the honest verifier in all sessions, except for session i. In session i, S^* honestly computes one of the temporary parameters $(k_e, t_e) \leftarrow \mathsf{PHTRec}(1^n, r)$ for a random bit e and sets the other parameter as $k_{\bar{e}} \leftarrow pk$. Then it forwards the pair k_0, k_1 to $\mathcal{P}^*$ and runs Σ^{trap} using (t_e, r) as witness. Note that messages provided by S^* are distributed identically to an honest verifier (and thus the extractor) therefore the behavior of $\mathcal{P}^*$ is not perturbed. Finally, as by assumption, it collects the first accepting transcript containing messages $\mathsf{tcom}_0, \mathsf{tcom}_1, \mathsf{tdec}_0, \mathsf{tdec}_1, s_0, s_1$ such that $s_0 \oplus s_1 = \mathsf{pok}_1^{\mathsf{L}_j}$. Then, S^* proceeds with the extraction phase exactly as the extractor [4]. Namely, it replicates the same transcript of the first phase but it replaces the message $\mathsf{pok}_2^{\mathsf{L}_j}$ with a fresh $\widetilde{\mathsf{pok}}_2^{\mathsf{L}_j}$ at each attempt. At the end of the extraction phase, as by assumption, S^* obtains $\mathsf{tcom}_0, \mathsf{tcom}_1, \widetilde{\mathsf{tdec}}_0, \widetilde{\mathsf{tdec}}_1, \tilde{s}_0, \tilde{s}_1$ such that $\tilde{s}_0 \oplus \tilde{s}_1 = \widetilde{\mathsf{pok}}_1^{\mathsf{L}_j}$ and $\mathsf{pok}_1^{\mathsf{L}_j} \neq \widetilde{\mathsf{pok}}_1^{\mathsf{L}_j}$ with non-negligible probability p/s. Since messages $\mathsf{pok}_1^{\mathsf{L}_j}, \widetilde{\mathsf{pok}}_1^{\mathsf{L}_j}$ are obtained from the xor of shares s_0, s_1 and $\tilde{s}_0, \tilde{s}_1$ respectively, one of the following two cases holds:

- Case 1. $s_{\bar{e}} \neq \tilde{s}_{\bar{e}}$: let $k_{\bar{e}} = pk$ be the parameter sent from the external receiver PHTRec. In this case, for the same commitment $\mathsf{tcom}_{\bar{e}}$ computed on $k_{\bar{e}} = pk$,

[4] Note that the extraction phase takes running time that is expected polynomial, while adversary S^* is strict polynomial. Therefore S^* will truncate the number of attempts if it exceeds a fixed polynomial. The fixed polynomial is set to $2 \cdot t_E$ where t_E is the expected running time of E. Since the case in which extraction requires super-polynomial number of attempts happens with negligible probability, the probability of success of S^* even truncating long executions is still non-negligible.

S^* has extracted from $\mathcal{P}^*$ two valid decommitments $\mathsf{tdec}_{\bar{\mathsf{e}}}, \widetilde{\mathsf{tdec}}_{\bar{e}}$ that open to two distinct strings: i.e., $\mathsf{tdec}_{\bar{\mathsf{e}}}$ opens to $s_{\bar{e}}$ and $\widetilde{\mathsf{tdec}}_{\bar{e}}$ opens to share $\tilde{s}_{\bar{e}}$ and $s_{\bar{e}} \neq \tilde{s}_{\bar{e}}$.

Hence, S^* can break the binding property of PHTCom with probability p/s as follows. In the commitment phase S^* sends $\mathsf{tcom}_{\bar{\mathsf{e}}}$ to PHTRec; in the decommitment phase it sends two valid decommitments $\mathsf{tdec}_{\bar{\mathsf{e}}}, \widetilde{\mathsf{tdec}}_{\bar{e}}$ to PHTRec. By the binding property of PHTCom Case 1 happens with negligible probability.

- Case 2. $s_e \neq \tilde{s}_e$ (and $s_{\bar{e}} = \tilde{s}_{\bar{e}}$): in this case S^* obtains two valid openings for the commitment $\mathsf{tcom}_{\mathsf{e}}$ that is computed with key k_e for which S^* knows the trapdoor t_e. In this case S^* does not break the binding. Note that trapdoor t_e has been used as witness to run protocol Σ^{trap}. Thus, if this case happens with non-negligible probability $p' = \frac{p}{s} - \nu(n)$ (p' corresponds to the probability $\frac{p}{s}$ minus the probability of Case 1, that we proved to be negligible), we have that $\mathcal{P}^*$ equivocates the commitment computed on the key for which $\mathcal{V}$ proves knowledge of the trapdoor using Σ^{trap}. Thus, we can construct an adversary $\mathcal{A}_{\mathsf{wi}}$ of the witness indistinguishability property of Σ^{trap}.

  $\mathcal{A}_{\mathsf{wi}}$ interacts with an external prover of Σ^{trap} that we denote by $\mathcal{P}^{\mathsf{trap}}$. $\mathcal{A}_{\mathsf{wi}}$ follows the procedure of S^* except that in session i it proceeds as follows. It computes parameters (k_0, t_0) and (k_1, t_1) of the trapdoor commitment scheme PHTCom by running the procedure $\mathsf{PHTRec}(1^n)$. It sends the statement $(k_0, k_1) \in L_{\mathsf{trap}}$ to the external prover $\mathcal{P}^{\mathsf{trap}}$ along with the pair of witnesses (t_0, t_1) and it forwards the messages received from $\mathcal{P}^{\mathsf{trap}}$ to $\mathcal{P}^*$. The other messages belonging to the protocol $\Pi_{\mathsf{c}\mathcal{Z}\mathcal{K}}$ or to concurrent sessions, are computed identically as S^*. Note that so far the simulation of S^* is perfect since the messages sent by $\mathcal{A}_{\mathsf{wi}}$ are distributed identically to the messages computed by S^*. In the extraction phase for session i, $\mathcal{A}_{\mathsf{wi}}$ follows the procedure S^* that in turns follows the extraction phase trying polynomially many rewinds (here, as above, we are assuming that the number of rewinding attempts is truncated to a fixed polynomial since $\mathcal{A}_{\mathsf{wi}}$ runs in strict polynomial time). Upon each extraction attempt, when simulating session i, $\mathcal{A}_{\mathsf{wi}}$ just copies the messages received by $\mathcal{P}^{\mathsf{trap}}$. Note that this is not a problem since in each rewind only the challenge $\mathsf{pok}_2^{\mathsf{L}_j}$ of session i is changed, while all messages sent *before*, are just copied. Since the last message of Σ^{trap} is sent along with challenge $\mathsf{pok}_2^{\mathsf{L}_j}$, the rewinding does not effect the external prover $\mathcal{P}^{\mathsf{trap}}$. Furthermore note that, since the temporary keys are independent in each sessions, $\mathcal{A}_{\mathsf{wi}}$ can simulate possibly concurrent sessions using new pairs of temporary keys (it does not need to query an external prover).

  When the extraction phase is completed, since $\mathcal{A}_{\mathsf{wi}}$ has played as S^*, it obtains two transcripts: $\{\mathsf{tcom}_0, \mathsf{tcom}_1, \mathsf{tdec}_0, \mathsf{tdec}_1, s_0, s_1\}$ and $\{\mathsf{tcom}_0, \mathsf{tcom}_1, \widetilde{\mathsf{tdec}}_0, \widetilde{\mathsf{tdec}}_1, \tilde{s}_0, \tilde{s}_1\}$ with probability p'. By assumption, we have that $\mathcal{P}^*$ provides two valid decommitments for the commitment computed with the key for which S^* has proved knowledge of the trapdoor. Namely, $\mathcal{P}^*$ opens the commitment $\mathsf{tcom}_{\mathsf{e}}$ as two distinct shares $s_e \neq \tilde{s}_e$ if trapdoor t_e has been used to prove theorem $(k_0, k_1) \in L_{\mathsf{trap}}$, and it opens the commitment $\mathsf{tcom}_{\bar{\mathsf{e}}}$ as two distinct shares $s_{\bar{e}} \neq \tilde{s}_{\bar{e}}$ when trapdoor

$t_{\bar{e}}$ has been used instead. Therefore, $\mathcal{A}_{\mathsf{wi}}$ outputs e if $s_e \neq \tilde{s}_e$ and $\bar{e}$ otherwise and its advantage of guessing the witness used by $\mathcal{P}^{\mathsf{trap}}$ is p'. By the witness indistinguishability property of Σ^{trap} we have that p' is negligible. Hence Case 2 happens with negligible probability.

Finally, recall that the extractor may abort also if 2^n attempts are made without obtaining another accepting transcript. We argue that such case happens with negligible probability as well.

Hence, E extracts the witness W with all but negligible probability, and the claim is proved. □

Claim 5.3.6. *The witness W extracted by E corresponds to w such that $(x, w) \in \mathcal{R}_L$, with overwhelming probability.*

Proof. Assume, towards a contradiction, that there exists a session i such that the witness W extracted at the end of the extraction phase is not a witness for $x \in L$, but is a tuple (v, d) for theorem $(\mathsf{com}_{\mathsf{op}}, \mathsf{pk}_j, \mathsf{tcom}_j) \in L_{\mathsf{op}_j}$. This means that $\mathcal{P}^*$ knows an opening (v, d) of $\mathsf{com}_{\mathsf{op}}$ such that v is a valid decommitment of tcom_j under the public key pk_j. Let p be the probability of such event.

We prove that p is negligible by hybrid arguments. In the following i is the session from which we extract the witness W.

H_0**:** This is the real game. $\mathcal{P}^*$ plays with an honest verifier (first phase of the extractor E). When extracting the witness for session i, E outputs the witness $W = (v, d)$ with probability p. (Recall that an honest verifier, with identity $j = (\mathsf{pk}_j, \mathsf{tcom}_j)$ runs the protocol Σ^{id_j} as following. It picks a random message m' and computes the opening of tcom_j to m' using the secret trapdoor sk and auxiliary information z. Such opening, that we denoted by $wdec$, is the witness used to complete the last round of protocol Σ^{id_j}.)

Given the extracted witness $W = (v, d)$, we have two cases:

Case 1: the opening v is a *new opening* of tcom_j. That is, $\mathcal{V}$ never used such opening as witness for Σ^{id_j} in any session.

Case 2: the opening v is equal to an opening used by $\mathcal{V}$ as witness for Σ^{id_j} in some other session. Namely, $v = (wdec_\ell, m_\ell)$, for some $\ell \in [s]$. In this case we can use $\mathcal{P}^*$ to distinguish the witness used in session ℓ. This reduction is very similar to the one shown in Hybrid H_1^ℓ below and therefore is omitted.

In the following sequence of hybrids we assume that we are in Case 1. Hence, we assume that the witness W extracted from $\mathcal{P}^*$ in H_0 is a *new opening* of tcom_j.

H_1^ℓ (for $1 < \ell < s$): In hybrid H_1^ℓ, in the first ℓ sessions $\mathcal{V}$ runs protocol Σ^{id_j} using the same witness, that is, the same opening of tcom_j. The remaining sessions are played according to the honest procedure, namely by using as witness freshly generated openings of tcom_j. Assume that the witness extracted in session H_1^ℓ

is not a new opening, but corresponds to one of the witnesses used by $\mathcal{V}$. Then we construct an adversary $\mathcal{A}_{\mathsf{wi}}$ for the witness indistinguishability of Σ^{id_j}. $\mathcal{A}_{\mathsf{wi}}$ works as follows. It computes the identity $(\mathsf{pk}_j, \mathsf{tcom}_j)$ and computes two openings $(m, wdec)$ and $(m', wdec')$. Then in the first $\ell - 1$ sessions, it runs Σ^{id_j} using as witness $(m, wdec)$, from session $\ell + 1$ on, it runs Σ^{id_j} using as witness freshly generated openings. To simulate the ℓ-th session, $\mathcal{A}_{\mathsf{wi}}$ uses the external prover of Σ^{id_j} fed with input $(\mathsf{pk}_j, \mathsf{tcom}_j)$, $(m, wdec), (m', wdec')$.

Once the first phase is completed $\mathcal{A}_{\mathsf{wi}}$ runs the extraction phase for session i. This basically means that $\mathcal{A}_{\mathsf{wi}}$ rewinds $\mathcal{P}^*$ up to the third round of session i, that is the point in which a new challenge $\widetilde{\mathsf{pok}}_2^{\mathsf{L}_j}$ is sent to $\mathcal{P}^*$. Note that, the new message can affect the messages played by $\mathcal{P}^*$ in concurrent sessions. In particular, care must be taken for session ℓ that was played by the external prover. This session can not be rewound. We have the two following cases.

1. The last message $\mathsf{pok}_3^{\mathsf{id}_j}$ of session ℓ is played *before* the message $\mathsf{pok}_2^{\mathsf{L}_j}$ of session i. In this case, the rewind does not affect session ℓ, since the transcript obtained up to this point is just copied from the previous phase. In this case, $\mathcal{A}_{\mathsf{wi}}$ extracts the witness W that by assumption is an opening for tcom_j. If such opening is still a new opening then we are in hybrid $H_1^{\ell-1}$, otherwise, if it is not a new opening, then we are in hybrid H_1^{ℓ}.
   To see why, recall that we are assuming that in $H_0 = H_1^1$, $\mathcal{P}^*$ provides an accepting transcript for session i and that the witness extracted is a new opening. Starting from H_1^1, the only difference between H_1^1 and H_1^2 is that, in H_1^2 the witness used in session 2 is the same as the one used in session 1. Now, if the witness extracted in H_1^2 is not a new opening, then one can use this fact to distinguish that session 1 and 2 use the same witness. The same holds for hybrids H_1^2 and H_1^3 and so on. Hence, by the witness indistinguishability property of Σ^{id_j}, hybrids H_1^{ℓ} and $H_1^{\ell-1}$ are indistinguishable.
2. The messages $\mathsf{pok}_2^{\mathsf{id}_j}, \mathsf{pok}_3^{\mathsf{id}_j}$ of session ℓ are played between the second message $\mathsf{pok}_2^{\mathsf{L}_j}$ and the third message $\mathsf{pok}_3^{\mathsf{L}_j}$ of the proof provided by $\mathcal{P}^*$ in session i . This means that, when $\mathcal{A}_{\mathsf{wi}}$ rewinds $\mathcal{P}^*$ and changes the message $\mathsf{pok}_2^{\mathsf{L}_j}$ of session i, this message could effect the message $\mathsf{pok}_2^{\mathsf{id}_j}$ that $\mathcal{P}^*$ sends in session ℓ. Since session ℓ was played by an external prover, $\mathcal{A}_{\mathsf{wi}}$ is stuck and fails in the extraction ($\mathcal{A}_{\mathsf{wi}}$ can not rewind the external prover).
   However, we argue that the witness used in session i can not depend on any concurrent session in which the message $\mathsf{pok}_3^{\mathsf{id}_j}$ (that is the only message of protocol Σ^{id_j} where the witness is used) is played *after* $\mathcal{P}^*$ sent $\mathsf{com}_{\mathsf{op}}$ in session i. The reason is the following. In the second round of session i, $\mathcal{P}^*$ is committing to the witness that it will be used to conclude Σ^{L_j}, without having seen the message $\mathsf{pok}_3^{\mathsf{id}_j}$ yet (recall that we are assuming that the witness extracted from $\mathcal{P}^*$ in session i is not for theorem $x \in L$). The witness used in $\mathsf{pok}_3^{\mathsf{id}_j}$ is a randomly chosen among a super-polynomial number of

possible witnesses. . Thus, the witness committed in $\mathsf{com_{op}}$ corresponds to the one played in $\mathsf{pok}_3^{\mathsf{id}_j}$ with negligible probability. Which means that $\mathcal{P}^*$ can maul the concurrent session ℓ only with a negligible probability. Thus the witness extracted will be independent of the one used in session ℓ with high probability.

However, the commitment $\mathsf{com_{op}}$ is only computationally binding. Thus, $\mathcal{P}^*$ could commit to some opening but still complete its proof using a witness that depends on the witness used in session ℓ. In this case, we have that $\mathcal{P}^*$ distinguishes and the way to detect it is to extract the witness, but $\mathcal{A}_{\mathsf{wi}}$ can not extract it. However, if this case happens with non-negligible probability it means the $\mathcal{P}^*$ is able to open the commitment $\mathsf{com_{op}}$ according to the witness played in the session ℓ. Such $\mathcal{P}^*$ can be used to break the binding of protocol PHCom.

Summing up, when the session ℓ (in which the witness used for Σ^{id_j} is changed), is interleaved with the session i (in which $\mathcal{P}^*$ convinces the verifier), the witness extracted in session i is independent of the one used in session ℓ except with negligible probability. Therefore, when changing the witness used in the session ℓ, the adversary $\mathcal{P}^*$ still proves session i by using a *new opening*.

In hybrid H_1^s, $\mathcal{V}$ runs protocol Σ^{id_j} using the same witness, i.e., the same opening of message tcom_j, in all sessions. Still, in H_1^s, $\mathcal{P}^*$ provides an accepting transcript for session i and the extracted witness is a new opening of tcom_j.

H_2: In this hybrid the commitment tcom_j is computed using the honest commitment procedure. That is, $(\mathsf{tcom}_j, wdec) \leftarrow \mathsf{TSen}(pk, m)$ instead of $(\mathsf{tcom}, z) \leftarrow \mathsf{TSen}(pk, \tilde{m})$. In each session, protocol Σ^{id_j} is run using $(wdec, m)$ as witness.

Assume towards a contradiction that in this hybrid the witness extracted in session i is *not* a new opening with non-negligible probability p. Then, we construct an adversary $\mathcal{A}_{\mathsf{trap}}$ for the trapdoorness of TCom that distinguishes whether a pair (commitment, decommitment) received from an oracle, is computed using the trapdoor or honestly. $\mathcal{A}_{\mathsf{trap}}$ computes parameters (pk, sk) and a message m and forwards them to the oracle. The oracle sends the challenge pair $(\mathsf{com}, \mathsf{dec})$ where com is the commitment of m under public key pk and dec is a valid opening. $\mathcal{A}_{\mathsf{trap}}$ uses $\mathcal{P}^*$ to distinguish this pair as follows. It publishes the identity $(\mathsf{pk}_j = pk, \mathsf{tcom}_j = \mathsf{com})$ and interacts with $\mathcal{P}^*$ as in hybrid H_1^s using the same witness (m, dec) to run protocol Σ^{id_j}. By contradicting assumption, in this hybrid, the witness extracted from $\mathcal{P}^*$ in session i is not a new opening of tcom_j with probability p. Thus, $\mathcal{A}_{\mathsf{trap}}$ extracts the witness of session i and if it does not corresponds to a new openings (or if $\mathcal{P}^*$ aborts) it outputs 1. Else it outputs a random bit. . By the trapdoorness property of TCom, the probability p is negligible.

Therefore in hybrid H_2 it holds that, $\mathcal{P}^*$ convinces $\mathcal{V}$ in session i with non-negligible probability p and the witness extracted in such session is still a new opening of

tcom_j. Given that, we construct another adversary S^* that breaks the binding of the scheme TCom as follows. S^*, running the binding experiment, obtains the public parameter pk. Then it picks a message m and computes $(\mathsf{tcom}_j, wdec) \leftarrow \mathsf{TSen}(pk, m)$ and publishes the identity $(\mathsf{pk}_j = pk, \mathsf{tcom}_j)$. It interacts with $\mathcal{P}^*$ as in hybrid H_2 and when the proof is completed it extracts the witness used by $\mathcal{P}^*$ in session i. By assumptions, with probability p this witness is a new opening of tcom_j, let us denote it by v. Thus, S^* outputs $(\mathsf{tcom}_j, wdec, m, v)$ and halts. By the binding property of the scheme TCom, the probability p is negligible.

Therefore we conclude that the probability that witness extracted by E is a valid opening of tcom_j instead of the witness w for $x \in L$ is negligible. □

5.3.2 Efficient Instantiation of $\Pi_{\mathsf{c}\mathcal{ZK}}$

In this section we describe how, by properly instantiating the sub-protocols of $\Pi_{\mathsf{c}\mathcal{ZK}}$, one can obtain a 4-round concurrently sound $\mathsf{c}\mathcal{ZK}$ in the BPK model based on the Discrete Logarithm (DL) assumption. We denote the efficient version of $\Pi_{\mathsf{c}\mathcal{ZK}}$ as $\bar{\Pi}_{\mathsf{c}\mathcal{ZK}}$. Let L be a language that admits a Σ-protocol Σ_L.

All commitment schemes used in the construction are instantiated with the two-round perfectly-hiding Pedersen commitment scheme [85]. In the first round the receiver computes the parameters as follows. It randomly chooses $(\mathbb{G}, p, q, g, h)$ such that $p = 2q + 1$, p and q are primes, $\mathbb{G}$ is a subgroup of order q of Z_p^* and g and h are generators of $\mathbb{G}$. Finally she sends parameters p, q, g, h to the committer, who checks that they are well formed. To commit to a message $m \in \mathbb{Z}_q$, the committer randomly picks $r \in \mathbb{Z}_q$ and sends $c = g^m h^r \pmod p$ to the receiver. The decommitment consists in revealing m, r.

To compute her identity, an honest verifier chooses $(\mathbb{G}, p, q, g, h)$ but h is chosen such that $h = g^\alpha \pmod p$ and she knows α. Then she randomly picks in $\mathbb{Z}_q$ two values r, m and computes $\mathsf{tcom}_j = g^m h^r \pmod p$. Finally she sets the public identity $\mathsf{pk}_j = (\mathsf{pk}_j = (\mathbb{G}, p, q, g, h), \mathsf{tcom}_j)$, while α, r, m are her secret input. $\mathcal{V}$ also generates and publishes parameters $(\mathbb{G}, p, q, g_L, h_L, g_R, h_R)$ where g_L, h_L, g_R, h_R are random generators of $\mathbb{G}$ as in the first round of Pedersen's scheme, to let then $\mathcal{P}$ compute $\mathsf{com}_{\mathsf{op}}$.

Protocol Σ^{id_j} run by $\mathcal{V}$ to prove knowledge of an opening of tcom_j is instantiated with the Σ-protocol Σ_{Ped} shown in Section 1.2.1.

The generation of the temporary keys for scheme PHTCom, corresponds to the execution of the first round of Pedersen's scheme. Hence, $k_0 = (\mathbb{G}_0, p_0, q_0, g_0, h_0)$, $k_1 = (\mathbb{G}_1, p_1, q_1, g_1, h_1)$, where $h_0 = g_0^{t_0} \pmod{p_0}$ and $h_1 = g_1^{t_1} \pmod{p_1}$. The corresponding trapdoors are t_0, t_1.

Protocol Σ^{trap} is used to prove knowledge of one of the trapdoors and can be instantiated using Schnorr [96] Σ-protocol under OR composition as discussed in [26] (for completeness such protocols are described in Section 1.2.1). Indeed, proving knowledge of one trapdoor corresponds to proving knowledge of the discrete logarithm of h_0 with base g_0 modulo p_0, or of the discrete logarithm of h_1 with base g_1 modulo p_1. Note that, in Pedersen's scheme knowledge of the trapdoor and the original opening, allows one to

equivocate any previously computed commitment (even without knowing the trapdoor during the commitment phase).

$\mathcal{P}$ computes commitment $\mathsf{com}_{\mathsf{op}}$ using parameter $(\mathbb{G}, p, q, g_L, h_R, g_R, h_R)$ stored in the public identity. Recall that $\mathsf{com}_{\mathsf{op}}$ should be such that it is possible to prove that the message committed in $\mathsf{com}_{\mathsf{op}}$ is an opening of tcom_j using efficient composition of Σ-protocols. Towards this end, $\mathsf{com}_{\mathsf{op}}$ is computed as follows.

Recall that $\mathsf{tcom}_j = g^m h^r \pmod p$. An opening of tcom is a pair (m, r). Thus, $\mathsf{com}_{\mathsf{op}}$ consists of a pair of commitments $\mathsf{com}^0_{\mathsf{op}}, \mathsf{com}^1_{\mathsf{op}}$ such that $\mathsf{com}^0_{\mathsf{op}} = g_L^m h_L^{s_0}$ for a random $s_0 \in \mathbb{Z}_q$ and $\mathsf{com}^1_{\mathsf{op}} = h_R^r g_R^{s_1}$ for a random $s_1 \in \mathbb{Z}_q$.

Now, in order to prove that $\mathsf{com}_{\mathsf{op}} = (\mathsf{com}^0_{\mathsf{op}}, \mathsf{com}^1_{\mathsf{op}})$ is a commitment of the opening of tcom_j, we exploit the fact that $\mathsf{com}_{\mathsf{op}}, \mathsf{tcom}_j$ are all Pedersen's commitments, and use Σ-protocol to prove equality of a pair of Pedersen's commitments as follows (such protocol is shown in Section 1.2.1 and is denoted by Σ_{Equal}). Informally, $\mathcal{P}$ proves that $\mathsf{com}^0_{\mathsf{op}}$ is the commitment of the message committed in tcom_j AND $\mathsf{com}^1_{\mathsf{op}}$ is the commitment of the randomness used in tcom_j.

Formally, $\mathcal{R}_{L^0_{\mathsf{Equal}}} = \{(\mathbb{G}, q, p, g_L, h_L, \mathsf{com}^0_{\mathsf{op}}, g, h, \mathsf{tcom}_j), (m, r, s_0) : \mathsf{tcom}_j = g^m h^r \pmod p$ and $\mathsf{com}^0_{\mathsf{op}} = g_L^m h_R^{s_0} \pmod p\}$. $\mathcal{R}_{L^1_{\mathsf{Equal}}} = \{(\mathbb{G}, q, p, h_R, g_R, \mathsf{com}^1_{\mathsf{op}}, g, h, \mathsf{tcom}_j), (r, m, s_1) : \mathsf{tcom}_j = h^r g^m \pmod p$ and $\mathsf{com}^1_{\mathsf{op}} = h_R^r g_R^{s_1} \pmod p\}$. Note that for this relation we swapped the generators, as in this case, the message we want to prove knowledge of, is the exponent of h instead of g. Hence, $\mathcal{R}_{L_{\mathsf{op}_j}} = \mathcal{R}_{L^0_{\mathsf{Equal}}} \wedge \mathcal{R}_{L^1_{\mathsf{Equal}}}$.

Finally, Σ^{L_j} can be instantiated as the OR composition of Σ-protocols Σ_L and Σ_{Equal} [26].

All above computations require a constant number of modular exponentiations. $\bar{\Pi}_{\mathsf{c}\mathcal{ZK}}$ is secure under the discrete logarithm assumption. All the Σ-protocols shown for the transformation are perfect.

Simulator Sim
Global variables. $h \leftarrow \emptyset$, $\mathcal{T} \leftarrow \emptyset$, $R_{\mathcal{V}^*} \xleftarrow{\$} \{0,1\}^{\mathtt{poly}(n)}$.

Main Thread.
Run $\mathcal{V}^*(R_{\mathcal{V}^*}, h)$ and answer to oracle queries in the form $(\mathsf{sid}, i, j, msg)$, depending on whether $msg = V1$ is the first round, or $msg = V3$ is the third round, as follows:

First Round: $V1 = (k_0, k_1, \mathsf{pok}_1^{\mathsf{id}_j}, \mathsf{pok}_1^{\mathsf{trap}})$

- **case identity j is already solved:** there exists $\mathtt{open}_j \in \mathcal{T}$ for tcom_j (in this case we say that session sid is solved).
  1. run step $\mathcal{P}$**-round-**2 and commit to $\mathtt{open}_j$ instead of a random string. Namely, $(\mathsf{com_{op}}, d) \leftarrow \mathsf{PHSen}(pk, \mathtt{open}_j)$ Let $P2 = (\mathsf{pok}_2^{\mathsf{id}_j}, \mathsf{pok}_2^{\mathsf{trap}}, \mathsf{com_{op}}, \mathsf{tcom_0}, \mathsf{tcom_1})$ the second round message.
  2. $h \leftarrow h \cup V1 \cup P2$;
  3. Run $\mathcal{V}^*(R_{\mathcal{V}^*}, h)$.
- **case identity j is not solved.** (in this case we say that session sid is not solved).
  1. pick random strings $\tilde{s}_0, \tilde{s}_1$ and compute: $(\mathsf{tcom_0}, \tilde{z}_0) \leftarrow \mathsf{PHTSen}(k_0, \tilde{s}_0)$, $(\mathsf{tcom_1}, \tilde{z}_1) \leftarrow \mathsf{PHTSen}(k_1, \tilde{s}_1)$;
  2. compute $\mathsf{com_{op}}$ as a commitment of a random string.
  3. let $m_{\mathsf{sid}} = \mathsf{com_{op}}, \mathsf{tcom_0}, \mathsf{tcom_1}$;
  4. let $h_{\mathsf{sid}} = h \cup V1$;
  5. let $P2 = (\mathsf{pok}_2^{\mathsf{id}_j}, \mathsf{pok}_2^{\mathsf{trap}}, \mathsf{com_{op}}, \mathsf{tcom_0}, \mathsf{tcom_1})$;
  6. $h \leftarrow h \cup V1 \cup P2$;
  7. Run $\mathcal{V}^*(R_{\mathcal{V}^*}, h)$.

Third Round: $V3 = (\mathsf{pok}_3^{\mathsf{id}_j}, \mathsf{pok}_3^{\mathsf{trap}}, \mathsf{pok}_2^{\mathsf{L}_j})$:

- **case sid is solved:**
  1. run step $\mathcal{P}$**-round-**4 but computes $\mathsf{pok}_3^{\mathsf{L}_j}$ using witness (v, d) and obtain: $P4 = (\mathsf{pok}_3^{\mathsf{L}_j}, \mathsf{tdec_0}, \mathsf{tdec_1}, s_0, s_1)$;
  2. $h \leftarrow h \cup V3 \cup P4$;
  3. Run $\mathcal{V}^*(R_{\mathcal{V}^*}, h)$;
- **case sid is not solved:** Sim needs to extract the temporary trapdoor:
  1. $t_e \leftarrow \mathsf{Extract}(\mathsf{sid}, h_{\mathsf{sid}}, m_{\mathsf{sid}})$, if the procedure Extract fails then `ABORT`;
  2. let $X = (x, \mathsf{com_{op}}, \mathsf{pk}_j, \mathsf{tcom}_j)$ the theorem to be proved with protocol Σ^{L_j}; compute $(\mathsf{pok}_1^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j}) \leftarrow \mathsf{hvSim}(X, \mathsf{pok}_2^{\mathsf{L}_j})$;
  3. $s_e \leftarrow \tilde{s}_{\bar{e}} \oplus \mathsf{pok}_1^{\mathsf{L}_j}$; $s_{\bar{e}} \leftarrow \tilde{s}_{\bar{e}}$;
  4. $\mathsf{tdec_e} \leftarrow \mathsf{PHTFakeDec}(t_e, \tilde{z}_e, s_e)$;
  5. $P4 \leftarrow (\mathsf{tdec_e}, \mathsf{tdec_{\bar{e}}}, \mathsf{pok}_3^{\mathsf{L}_j}, s_0, s_1)$;
  6. $h \leftarrow h \cup V3 \cup P4$;
  7. Run $\mathcal{V}^*(R_{\mathcal{V}^*}, h)$.

Exit. If $\mathcal{V}^*$ has completed its attack, or if it aborts then output h. In case $\mathcal{V}^*$ sends invalid messages, add those messages to h and output it.

Figure 5.5: *Simulator associated to protocol* $\Pi_{\mathsf{c}\mathcal{ZK}}$.

Procedure $\mathsf{Extract}(\mathsf{sid}, h_{\mathsf{sid}}, m_{\mathsf{sid}})$
(sid, i, j) is the target session for which Sim aims to extract the trapdoor t_e.
$\mathsf{sid}' \neq \mathsf{sid}$ denotes any other session different from the target session sid.

Repeat at most 2^n **times.**
Start.
- Pick $\widetilde{\mathsf{pok}}_2^{\mathsf{id}_j}, \widetilde{\mathsf{pok}}_2^{\mathsf{trap}}$ using fresh randomness.
- Set $\tilde{h}_{\mathsf{sid}} \leftarrow h_{\mathsf{sid}} \cup (m_{\mathsf{sid}}, \widetilde{\mathsf{pok}}_2^{\mathsf{id}_j}, \widetilde{\mathsf{pok}}_2^{\mathsf{trap}})$.
- Run $\mathcal{V}^*(R_{\mathcal{V}^*}, \tilde{h}_{\mathsf{sid}})$. If $\mathcal{V}^*$ aborts, go to **Start**. Else, answer to the message M received from $\mathcal{V}^*$ as follows:

Case 1. If $M = (\mathsf{sid}', i', j', V1)$: follow step First Round of Sim without updating h.

Case 2. If $M = (\mathsf{sid}', i', j', V3)$: if sid' is *unsolved*, solve this session. Else, follows instructions of Sim.

Case 3. If $M = (\mathsf{sid}, i, j, \widetilde{V}3 = (\widetilde{\mathsf{pok}}_3^{\mathsf{id}_j}, \widetilde{\mathsf{pok}}_3^{\mathsf{trap}}, \mathsf{pok}_2^{\mathsf{L}_j}))$:

1. if $(\widetilde{\mathsf{pok}}_3^{\mathsf{trap}} \neq \mathsf{pok}_3^{\mathsf{trap}})$ extract witness t_e;
2. if $(\widetilde{\mathsf{pok}}_3^{\mathsf{id}_j} \neq \mathsf{pok}_3^{\mathsf{id}_j})$ extract witness $\mathtt{open}_j$; add $\mathcal{T} \leftarrow \mathcal{T} \cup \mathtt{open}_j$; return t_e;
3. else abort.

If $\mathcal{V}^*$ sends invalid messages or aborts the session, go to **Start**.

Case exit. If number of attempts is 2^n, output $\perp$ and halt.

Figure 5.6: *The procedure* Extract.

Extractor E.

Setup Identity. Run $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{TGen}(1^n)$ and $(\mathsf{tcom}_j, z) \leftarrow \mathsf{TSen}(\mathsf{pk}_j, \tilde{m})$. Publish $j = (\mathsf{pk}_j, \mathsf{tcom}_j)$.

First Phase (honest verifier phase). Choose random tapes $R_{\mathcal{P}^*}$, $R_{\mathcal{V}}$. Activate $\mathcal{P}^*$ with $R_{\mathcal{P}^*}$ and run the honest verifier procedure using $R_{\mathcal{V}}$ and sk_j, z. After $\mathcal{P}^*$ halts, if there exists an accepting session i for some statement "$x \in L$" chosen by $\mathcal{P}^*$, then go to the extraction phase. Else, output $\perp$ and halt.

Extraction Phase. Let i be a randomly picked session among sessions that were accepting, and let $\{\mathsf{pok}_1^{\mathsf{L}_j} = (s_0 \oplus s_1), \mathsf{pok}_2^{\mathsf{L}_j}, \mathsf{pok}_3^{\mathsf{L}_j}\}$ be the transcript of such session. Here s_0 and s_1 are the valid openings of trapdoor commitments $\mathsf{tcom}_0, \mathsf{tcom}_1$ sent by $\mathcal{P}^*$ in the second round of session i.
Repeat until the extraction is successful or until 2^n attempts fail.
Next Attempt. Activate $\mathcal{P}^*$ with $R_{\mathcal{P}^*}$.
For all sessions except session i run as in first phase (i.e., as honest verifier) with randomness $R_{\mathcal{V}}$. In session i, compute messages as in first phase but pick a new challenge $\widetilde{\mathsf{pok}}_2^{\mathsf{L}_j}$. If $\mathcal{P}^*$ does not reply with a valid third message, go to Next Attempt. Else: If $\mathcal{P}^*$ successfully terminates session i again, with transcript $\{\widetilde{\mathsf{pok}}_1^{\mathsf{L}_j} = (\tilde{s}_0 \oplus \tilde{s}_1), \widetilde{\mathsf{pok}}_2^{\mathsf{L}_j}, \widetilde{\mathsf{pok}}_3^{\mathsf{L}_j}\}$ and $\widetilde{\mathsf{pok}}_1^{\mathsf{L}_j} \neq \mathsf{pok}_1^{\mathsf{L}_j}$ then `ABORT`. (In this case $\mathcal{P}^*$ was able to open at least one between tcom_0 and tcom_1 as a distinct string with respect to the one obtained in the first phase). Otherwise, use the new transcript to extract the witness W for the relation $\mathcal{R}_{L_{L_j}}$. Output W and halt.
If 2^n attempts failed, then `ABORT`.

Figure 5.7: *The extractor E for protocol* $\Pi_{\mathsf{c}\mathcal{Z}\mathcal{K}}$.

Part

Security in Presence of Reset Attacks

- Chapter 6 -

Simultaneously Resettable Arguments of Knowledge

Introduction

Interaction and private randomness are the two fundamental ingredients in Cryptography. They are especially important for achieving zero-knowledge proofs [51]. In [19] Canetti, Goldreich, Goldwasser and Micali showed that when private randomness is limited and re-used in multiple instances of a proof system, it is still possible to preserve the zero-knowledge requirement. The setting proposed by [19] is of a malicious verifier that resets the prover, therefore forcing the prover to run several protocol executions using the same randomness. This setting applies to protocols where the prover is implemented by a stateless device. Therefore, a prover can only count on the limited (hardwired) randomness while it can be adaptively reset any polynomial number of times. The resulting security notion against such powerful verifiers is referred to as *resettable zero knowledge* ($\mathsf{r}\mathcal{ZK}$) and is provably harder to achieve than concurrent zero knowledge. Feasibility results have been achieved in [19, 65] in the standard model with the following round-complexity: polylogarithmic for $\mathsf{r}\mathcal{ZK}$ and constant for resettable witness indistinguishability ($\mathsf{r}\mathcal{WI}$, in short). Since then, it was also shown how to achieve resettable zero knowledge in the Bare Public-Key (BPK) model, introduced by Canetti et al. [19], where one can obtain better round complexity and assumptions [73, 31, 1, 106, 95]. Very recently, it has been shown [45] that resettable statistical zero knowledge for non-trivial languages is possible.

The "reverse" of the above question has been considered by Barak, Goldreich, Goldwasser and Lindell [7] where a malicious prover resets a verifier, called *resettable soundness*. In [7], it has been shown how to obtain resettable soundness along with $\mathcal{ZK}$ in a constant number of rounds.

Barak et al. [7] proposed the challenging *simultaneous resettability conjecture*, where one would like to prove that a protocol is secure against both a resetting malicious prover and a resetting malicious verifier. The existing machinery turned out to be insufficient, and a definitive answer required almost a decade. In the work of Deng, Goyal and Sahai [29] they showed a resettably sound $\mathsf{r}\mathcal{ZK}$ argument for **NP** with polynomial round

complexity. Very recently, results in the BPK model for simultaneous resettability have been obtained in [105, 2] with a constant number of rounds.

Arguments of knowledge under simultaneous resettability. Argument systems are often used with a different goal than proving membership of an instance in a language. Indeed, it is commonly required to prove knowledge (possession) of a witness instead of the truthfulness of a statement. Since arguments of knowledge serve as major building blocks in Cryptography (e.g., in identification schemes[1]), it is an interesting question whether the previous results for arguments of membership extend to arguments of knowledge. Unfortunately, arguments of knowledge have been achieved so far only when one party can reset. That is, we have $\mathsf{r}\mathcal{ZK}$ arguments of knowledge [19] and, separately, resettably sound $\mathcal{ZK}$ arguments of knowledge [7]. Instead, when reset attacks are possible in both directions, no result is known even when only $\mathsf{r}\mathcal{WI}$ with resettable argument of knowledge is desired.

It is important to note that resettable security for ZAPs comes almost for free because of the minimal round complexity (1 or 2 rounds). However, it is not known how to accommodate for knowledge extraction, unless one relies on non-standard (e.g., non-falsifiable) assumptions. For the case of resettably sound $\mathsf{r}\mathcal{ZK}$, all the above results [29, 105, 2] critically use an instance-dependent technique along with ZAPs: when the statement is true (i.e., when proving $\mathsf{r}\mathcal{ZK}$), the prover/simulator can run ZAPs which allow the use of multiple witnesses. Such use of multiple witnesses gives some flexibility that turns out to be very useful to prove resettable zero knowledge. Instead, when the statement is false, the protocols are designed so that adversarial malicious prover must stick with some fixed messages during the execution of protocol. Therefore, rewinding capabilities do not help the resetting malicious prover since he can not change those fixed messages. This is critically used in the proofs of resettable soundness in order to reach a contradiction when a prover proves a false statement. It is easy to see that the above approach fails when arguments of knowledge are considered. Indeed, when the malicious resetting prover proves a true statement, the same freedom that allows one to prove $\mathsf{r}\mathcal{ZK}/\mathsf{r}\mathcal{WI}$, also gives extra power to the malicious prover. Consequently, designing an extractor appears problematic and new techniques seem to be needed so that the simultaneous resettability conjecture is resolved even when we consider knowledge extraction.

Our main result is the first construction of a constant-round simultaneously resettable witness-indistinguishable argument of knowledge[2] (in short, $\mathsf{simres}\mathcal{WI}\mathsf{AoK}$) for any **NP**

[1] Bellare et al. in [9] gave various definitions for identification schemes when the adversary can also reset the proving device.

[2] In this work, we will never consider the case of resettable soundness along with non-resettable argument of knowledge. Therefore, each time we mention together resettable soundness and argument of knowledge, we mean that both soundness and witness extraction hold against a malicious resetting prover.

language. Our protocol is based on the novel use of ZAPs and resettably sound zero-knowledge arguments, which improves over the techniques previously used in [29, 105] as well as concurrent and independent work[3] of [56].

As application of our main protocol, we also consider the question of secure identification under simultaneous resettability and show how to use the above simres$\mathcal{WI}$AoK to obtain the first simultaneously resettable identification scheme which follows the knowledge extraction paradigm. We describe it by extending the work of Bellare, et al. [9].

This chapter is organized as follows. In Section 6.1 the definition of simultaneously resettable argument of knowledge is formally stated. In Section 6.2 the construction for simultaneous resettable WI argument of knowledge (simres$\mathcal{WI}$AoK, in short), along with formal proofs is presented. Finally, Section 6.3 shows the application of our construction to identification schemes.

6.1 Defintions

This section provides formal definitions of resettable $\mathcal{WI}$ and resettable soundness. In particular, we formally define the notions of resettable argument of knowledge.

6.1.1 Resettable Zero Knowledge and Witness Indistinguishability.

We recall the definition of resettable zero knowledge and witness indistinguishability introduced in [19]. Very roughly, a resetting verifier is a PPT adversary that is able to interact with the prover polynomially many times upon (possibly) distinct theorems forcing the prover to execute the protocol using the same randomness several times. Namely, the malicious verifier invokes the prover by two indexes (i, j): the theorem to be proved x_i and the randomness to be used ω_j. The formal definition is provided below and is taken almost verbatim from [19].

Definition 25 (r$\mathcal{ZK}$ and r$\mathcal{WI}$ [19])**.** *An interactive proof system $(\mathcal{P}, \mathcal{V})$ for a language L is said to be* `resettable zero knowledge` *(r$\mathcal{ZK}$) if for every PPT adversary $\mathcal{V}^*$ there exists a probabilistic polynomial time simulator* M *so that distribution ensembles D_1 and D_2 described below are computationally indistinguishable. Let each distribution be indexed by a sequence of distinct common inputs $\bar{x} = x_1, \ldots, x_{\mathtt{poly}(n)} \in L \cap \{0,1\}^n$ and the corresponding prover's auxiliary-inputs $\bar{y} = y_1, \ldots, y_{\mathtt{poly}(n)}$.*

`Distribution` *D_1 is defined by the following random process which depends on $\mathcal{P}$ and $\mathcal{V}^*$.*

Randomly select and fix $t =$ `poly`(n) random tapes $\omega_1, \ldots, \omega_t$ for $\mathcal{P}$, resulting in deterministic strategies $\mathcal{P}^{(i,j)} = \mathcal{P}_{x_i, y_i, \omega_j}$ defined by $\mathcal{P}_{x_i, y_i, \omega_j}(\alpha) =$

[3] In a very recent and independent work [56], Goyal and Maji achieved simultaneously resettable secure computation. Their work achieves (with simulation-based security) simultaneous resettability with polynomial round complexity assuming also the existence of lossy trapdoor encryption.

$\mathcal{P}(x_i, y_i, \omega_j, \alpha)$ *for* $j \in 1, \ldots, t$, $i \in 1, \ldots, \texttt{poly}(n)$. *Each of the execution* $\mathcal{P}^{(i,j)}$ *it is called* `incarnation` *of* $\mathcal{P}$.

1. *Machine* $\mathcal{V}^*$ *is allowed to run polynomially-many sessions with* $\mathcal{P}^{(i,j)}$. *Throughout these sessions,* $\mathcal{V}^*$ *is required to complete its current interaction (an interaction is complete if is either terminated or aborted) with the current copy of* $\mathcal{P}^{(i,j)}$ *before starting a new interaction with any* $\mathcal{P}^{(i',j')}$, *regardless if* $(i,j) = (i',j')$ *or not. Thus, the activity of* $\mathcal{V}^*$ *proceeds in rounds. In each round it selects a* $\mathcal{P}^{(i,j)}$ *and conducts a complete executions with it.*
2. *Once* $\mathcal{V}^*$ *decides it is done interacting with all* $\mathcal{P}^{(i,j)}$, *it produces an output based on its view of these interactions. This output is denoted by* $\langle \mathcal{P}(\bar{y}), \mathcal{V}^* \rangle(\bar{x})$ *and is the output of the process.*

`Distribution` D_2`:` *The output of* $\mathsf{M}(\bar{x})$.

An interactive proof system $(\mathcal{P}, \mathcal{V})$ *for a language* L *is said to be* `resettable witness indistinguishable` *(*rWI*) if every two distribution ensembles of type* D_1 *that are indexed by the same sequence of distinct inputs* $\bar{x} = x_1, \ldots, x_{\texttt{poly}(n)} \in L \cap \{0,1\}^n$ *but possibly different sequences of prover's auxiliary inputs:* $\bar{y}^{(0)}(\bar{x}) = y_1^0, \ldots, y_{\texttt{poly}(n)}^0$ *and* $\bar{y}^{(1)}(\bar{x}) = y_1^1, \ldots, y_{\texttt{poly}(n)}^1$ *are computationally indistinguishable. That is, we require that ensembles* $\{\langle \mathcal{P}(\bar{y}^{(0)}), \mathcal{V}^* \rangle(\bar{x})\}_{\bar{x}}$ *and* $\{\langle \mathcal{P}(\bar{y}^{(1)}), \mathcal{V}^* \rangle(\bar{x})\}_{\bar{x}}$ *are computationally indistinguishable.*

Resettable Soundness. In the following definition we consider only computationally bounded malicious provers.

Definition 26 (resettable soundness rs [7])**.** *A resetting attack of a cheating prover* $\mathcal{P}^*$ *on a resettable verifier* $\mathcal{V}$ *is defined by the following two-step random process, indexed by a security parameter* n.

1. *Uniformly select and fix* $t = \texttt{poly}(n)$ *random tapes, denoted* $r_1, \ldots, r_t$, *for* $\mathcal{V}$ *resulting in deterministic strategies* $\mathcal{V}^{(j)}(x) = \mathcal{V}_{x,r_j}$ *defined by* $\mathcal{V}_{x,r_j}(\alpha) = \mathcal{V}(x, r_j, \alpha)$, *where* $x \in \{0,1\}^n$ *and* $j \in 1, \ldots, t$. *Each* $\mathcal{V}^{(j)}(x)$ *is called an* `incarnation` *of* $\mathcal{V}$.
2. *On input* 1^n, *machine* $\mathcal{P}^*$ *is allowed to initiate* $\texttt{poly}(n)$*-many interactions with* $\mathcal{V}$. *The activity of* $\mathcal{P}^*$ *proceeds in rounds. In each round* $\mathcal{P}^*$ *chooses* $x \in \{0,1\}^n$ *and* $j \in 1, \ldots, t$, *thus defining* $\mathcal{V}^{(j)}(x)$, *and conducts a complete session (again, a session is complete if is either terminated or aborted) with it.*

Let $\mathcal{P}$ *and* $\mathcal{V}$ *be some pair of interactive machines, where* $\mathcal{V}$ *is implementable in probabilistic polynomial-time. We say that* $(\mathcal{P}, \mathcal{V})$ *is a resettably-sound argument system (*rs*) for* L *if the following two conditions hold:*

- Resettable-completeness: *Consider a polynomial-size resetting attack and suppose that in some session after selecting an incarnation* $\mathcal{V}^{(j)}(x)$, *the attacker follows the strategy* $\mathcal{P}$ *(for those sessions will also be given the witness). Then, if* $x \in L$ *then* $\mathcal{V}^{(j)}(x)$ *rejects with negligible probability.*

- Resettable-soundness: *For every polynomial-size resetting attack, the probability that in some session the corresponding $\mathcal{V}^{(j)}(x)$ has accepted and $x \notin L$ is negligible.*

Arguments of knowledge in the simultaneous resettability setting.

Proving the argument of knowledge property of an argument system usually requires to show an expected polynomial-time Turing Machine called Extractor, that having oracle access to the prover, is able to extract the witness of any accepting proof. Such extractor is called black-box extractor. In presence of a resetting verifier, where $\mathcal{V}^*$ is allowed to rewind the honest prover, the verifier has the same power of the extractor, therefore a protocol cannot be resettable $\mathcal{WI}$ (or $\mathsf{r}\mathcal{ZK}$) and proof (or argument) of knowledge at the same time, unless we provide the extractor with some additional power respect to the malicious verifier. Thus as a natural relaxation of the standard notion of proof of knowledge [10] to the simultaneously resettable setting, we consider non-black box extraction, that is, the extractor gets the code of the (possible malicious) prover.

Definition 27 (resettably-sound argument of knowledge (adapted from [7]))**.** *Let $\mathcal{R}_L : \subseteq \{0,1\}^* \times \{0,1\}^*$ be an* **NP***-relation for a language $L = \{x : \exists y \ (x,y) \in \mathcal{R}_L\}$. We say that $(\mathcal{P}, \mathcal{V})$ is a* resettably-sound argument of knowledge *for $\mathcal{R}_L$ if:*

- *$(\mathcal{P}, \mathcal{V})$ is a resettably-sound argument for L;*
- *there exists an expected polynomial time extractor* **E** *such that for every PPT resetting machine $\mathcal{P}^*$, there exists a negligible function ϵ for which the following condition holds:*

$$|\mathsf{Pr}\,[\,(\mathcal{P}^*, \mathcal{V})(x) = 1\,] - \mathsf{Pr}\,[\,\mathbf{E}(desc(\mathcal{P}^*), x) \in \mathcal{R}_L(x)\,]\,| < \epsilon(|x|)$$

where $desc(\mathcal{P}^*)$ denotes the description of $\mathcal{P}^*$'s strategy and $\mathcal{R}_L(x)$ denotes the set of witnesses for x for the **NP** *language L.*

Remark 3 (Our Extractor)**.** *The extractor that we provide in our main construction (the simultaneous resettable WI argument of knowledge) follows the typical two-phase extraction paradigm: in the first phase the extractor runs the honest verifier procedure, and if it obtains an accepting transcript, proceeds to the second phase. In the second phase, it resets $\mathcal{P}^*$ several times and simulates the verifier using fresh randomness trying to reconstruct a new accepting transcript that allows for extraction.*

In order to use the extractor in the security proof of a larger protocol, we define our extractor with three random tapes in input. The randomness used to activate $\mathcal{P}^*$, the randomness used to perform the first phase (the length of these tapes is polynomially bounded by the size of the prover), and the one used for the second phase (the length of this last tape can not be established in advance since the number of attempts required to obtain a distinct accepting transcript is not dependent of the size of $\mathcal{P}^*$).

6.1.2 Resettably Sound Statistical Zero Knowledge.

In our construction we will use the constant-round public-coin zero-knowledge argument of Barak [5, 7] as modified in [83]. In [83] the original Barak's protocol is tweaked in order to obtain statistical zero knowledge (instead of computational ZK) for **NP** languages along with argument of knowledge (instead of weak proof of knowledge). Finally by applying the transformation of [7] to the construction of [83] one can obtain a resettably-sound statistical zero knowledge argument of knowledge.

ZAPs [38, 57]. ZAPs are two-round resettably-sound witness indistinguishable proof systems [38]. As noted in [38] by requiring that the randomness of the prover is generated by applying a pseudo-random function to the first message sent by the verifier, ZAPs are also resettable witness indistinguishable. We refer to such a simultaneously resettable ZAP as rZAP.

6.2 Simultaneously Resettable Arguments of Knowledge

Our goal is to obtain a construction that is resettably-sound resettable $\mathcal{WI}$ *and* a resettable argument of knowledge in a constant number of rounds. The only known constant-round simultaneously-resettable $\mathcal{WI}$ protocol is rZAP which is not an argument of knowledge and as discussed previously there is not much hope to transform it in an argument of knowledge (even without considering resettability).

A typical paradigm: determining message and consistency proof. Typically, protocols dealing with a resetting adversary ([19, 7, 29]) rely on the following paradigm: the resetting party is required to provide a special message (called *determining message*) that determines her own action for the rest of the protocol. Namely, for each protocol message the resetting party is required to prove that such message is consistent with the determining message (we call this proof a *consistency proof*). Moreover, the actual randomness used by the honest party in the protocol depends on the determining message (typically the honest party applies a pseudorandom function (PRF) on it). The combination of the randomness depending on the determining message and the consistency proof given by the resetting party, suppresses the resetting power of the adversary. Indeed, due to the consistency proof, the resetting party can not change a message previously played without first having changed the determining message (unless she is able to fake the consistency proof). However, if she changes the determining message, then the honest party plays the protocol with (computationally) fresh randomness (unless the pseudo-randomness of the PRF is violated). We will follow this paradigm to construct our simultaneously resettable witness indistinguishable argument of knowledge as well. Recall that as specified above, we do not know how to from rZAPs that are already simultaneously resettable and try to transform them in arguments of knowledge. Our starting point is Blum's proof of knowledge [12]. In the following discussion we show incrementally how to transform such protocol to enjoy resettable witness indistinguisha-

bility and resettable soundness (this transformation is already known in literature) to finally present our novel technique to obtain also *resettable* argument of knowledge.

Resettable $\mathcal{WI}$ and stand-alone argument of knowledge [7]. When the verifier can reset the prover, following the above paradigm, it is easy to construct a resettable $\mathcal{WI}$ system starting from Blum's protocol. In Blum's protocol the only message from $\mathcal{V}$ to $\mathcal{P}$ is the challenge. The modified resettable version requires that $\mathcal{V}$ sends a statistically binding commitment of the challenge as determining message. The only other protocol message of $\mathcal{V}$ is the opening of the commitment which, due to the binding property, is itself a proof that the message is consistent with the determining message. Note that such modified protocol is no longer an argument of knowledge since the extractor has the same power of the malicious verifier. In order to allow only the extractor to cheat, the next step is to avoid the opening as a proof of consistency. Instead of the actual opening of the commitment, $\mathcal{V}$ is required to send the challenge along with a res-sound (non-black-box) $\mathcal{ZK}$ argument ([5]). The (non-black box) extractor can send an arbitrary challenge and prove consistency with the determining message by using the (stand-alone) non-black-box simulator (recall that only $\mathcal{V}$ might reset here). The resulting protocol is resettable $\mathcal{WI}$ and (stand-alone) argument of knowledge (r$\mathcal{WI}$AoK for short) and it is known from [7].

We use a modified version of such protocol. We require that the commitment sent by the verifier is statistically hiding (instead of statistically binding), and we use the statistical zero-knowledge argument of knowledge of [83].

Achieving Resettable Soundness and Resettable Argument of Knowledge: existent solutions do not work. We now deal with the case in which also the prover can reset. By the BGGL compiler [7], we know that any constant-round public-coin $\mathcal{WI}$ argument system can be upgraded to resettable soundness by simply requiring the honest verifier to apply a PRF on the first message received from the prover. However, since our aim is to obtain simultaneous resettability, we need to start from the r$\mathcal{WI}$AoK protocol shown before, which is not public coin. Thus, following the paradigm and the technique of [29], we require that as first message, $\mathcal{P}$ sends the commitment of the randomness that will be used in the protocol: this is the determining message. Then upon each protocol message $\mathcal{P}$ proves that the message is honestly computed using the randomness committed in the determining message: this is the consistency proof. Since we are now in the setting in which both parties can reset each other the consistency proof must be provided with a simultaneous resettable tool. For this purpose we use rZAPs that are constant-round simultaneously resettable $\mathcal{WI}$ proofs. We denote the theorem to be proved with rZAP as "consistency theorem", since $\mathcal{P}$ proves that a message is honestly computed and consistent with the randomness committed in the determining message.

The technical problem using rZAPs is that since guarantee $\mathcal{WI}$, the theorem being proved is required to have more than one witness (note that the simultaneously resettable protocol of [29] can not be used here since we aim to a constant-round construction). Recall that we want to use rZAP to provide the proof of consistency with the

determining message. If the determining message is a statistically binding commitment of the randomness, then there exists a unique opening, which implies the existence of only one witness. On the other hand, if we use a statistically hiding commitment, then any opening is a legitimate witness, the theorem is always true and the benefit of the determining message vanishes. The solution to overcome this problem is to change the theorem to be proved with rZAP so that it admits more than one witness.

In [29] the consistency theorem is augmented with the theorem "$x \in L$" that we call "*trapdoor* theorem" recalling FLS paradigm [42] but with a different purpose. We call it trapdoor to stress out that it is an escape for the prover that can pass the consistency proof essentially having freedom to change messages among resets. Hence in [29, 105], along with each protocol message, $\mathcal{P}$ is required to prove that either the protocol message is computed honestly with the randomness committed in the determining message, (i.e., the "consistency theorem") or $x \in L$ (i.e., the "trapdoor theorem").

This solution can be seen as an instance-dependent technique. Indeed, it is easy to see that a malicious prover can play messages inconsistently with the determining message and still pass the consistency check, therefore exploiting its resetting power, only when $x \in L$. Instead, when proving soundness, since $x \notin L$, the trapdoor theorem is false, hence due to soundness of rZAPs, the malicious prover is forced to play according to the determining message therefore honestly following the protocol specifications.

Unfortunately, such an instance-dependent solution suffices to prove resettable soundness but fails completely when one would like to prove witness extraction (i.e., the argument of knowledge property). The reason is that, when proving witness extraction, we have to construct an extractor that works against any malicious prover, even one who uses the witness of the trapdoor theorem when proving consistency of the protocol messages. This possible behavior harms the extractor in two ways (recall that the witness can be computed from two distinct transcripts of Blum's protocol that have the *same* first message): 1) upon seeing the challenge of the verifier/extractor, $\mathcal{P}$ resets it and changes the first message of Blum's protocol according to the challenge; 2) $\mathcal{P}$ acts as a resetting verifier in the non-black-box $\mathcal{ZK}$ protocol, therefore preventing the extractor to use the stand-alone non-black-box simulator. Even though this is not harmful for the soundness property (a malicious prover can perform this attack only when $x \in L$), this attack kills the existence of the extractor. Therefore the above construction is only resettable $\mathcal{WI}$ and resettable sound. Concluding, the instance-dependent technique of [29] inherently prevents the existence of any extractor. New ideas are required to solve the problem.

Achieving *Resettable* Argument of Knowledge: the new technique. We propose a new "trapdoor" theorem that forces the resetting prover to honestly follow the protocol regardless of whether $x \in L$ or not.

The idea is the following. We require $\mathcal{P}$ to run *two* parallel executions of the r$\mathcal{WI}$AoK shown above, that we denote as subprotocols Π_0,Π_1. In the determining message, in addition to the commitment of the random tape that will be used to run each sub-protocol, we require that $\mathcal{P}$ commits to a single bit. Then, the trapdoor theorem in sub-protocol

Π_d will be the following: "d is the bit committed in the determining message". Since in the determining message there is only one bit committed (the other two are commitments of random tapes), due to the statistical binding property of the commitment, the trapdoor theorem is true in only one sub-protocol. Hence, in at least one of the sub-protocols the trapdoor theorem is false regardless of whether $x \in L$ or not, and in such sub-protocol $\mathcal{P}$ is forced to honestly follow the r$\mathcal{WI}$AoK protocol, playing consistently with the determining message.

More specifically, the final protocol goes as follows. $\mathcal{P}$ first sends the determining message which consists of the statistically binding commitment of the random tapes that will be used in each sub-protocol and of a single bit. Each sub-protocol is augmented with rZAPs sent by $\mathcal{P}$ to $\mathcal{V}$ in which $\mathcal{P}$ proves consistency with the determining message. Therefore, in each sub-protocol Π_d, along with each message of the r$\mathcal{WI}$AoK protocol, $\mathcal{P}$ provides a rZAP for the following compound theorem: either the message is honestly computed and consistent with the determining message, or d is the bit committed in the determining message. Finally, the verifier will accept the proof if and only if *both* sub-protocol executions are accepting.

It is easy to see that any malicious prover can not escape from following the determining message in at least one of the subprotocols. Indeed, let b be the bit committed in the determining message. If on one hand, in sub-protocol Π_b, a malicious $\mathcal{P}$ is not forced to be honest and can then use the resetting power to prove any false theorem (indeed among resets $\mathcal{P}$ can change the protocol messages without changing the determining message), on the other hand, in sub-protocol $\Pi_{\bar{b}}$, the trapdoor theorem is false, thus the only way to provide an accepting rZAP is to follow the honest behavior playing messages derived from the determining message. Therefore, in sub-protocol $\Pi_{\bar{b}}$, the extractor is guaranteed that 1) for sessions starting with the same determining message, the first round of Blum's protocol does not change, so that playing with two distinct challenges yields the extraction of the witness; 2) the extractor can run the stand-alone non-black-box $\mathcal{ZK}$ simulator without being detected. Hence we have the following: sub-protocol $\Pi_{\bar{b}}$ is resettably-sound and resettable argument of knowledge, while sub-protocol Π_b is not sound. Note that in both sub-protocols, the resettable $\mathcal{WI}$ property is still preserved.

6.2.1 Construction of simres$\mathcal{WI}$AoK

We formally describe how to build a constant-round simres$\mathcal{WI}$AoK starting from Blum's protocol (BL protocol). We denote by SHCom, a two-round statistically hiding commitment scheme. We denote by SBCom the commitment procedure of a non-interactive statistically binding commitment scheme. We denote by $c \leftarrow \mathsf{SBCom}(v, s)$ (resp. SHCom) the output of the commitment of the value v computed with randomness s. We use the resettably-sound statistical (non-black-box) $\mathcal{ZK}$ AoK of [83] that we denote by resSZK. In our construction, we require that $\mathcal{P}$, at each round of the protocol (except the last that is the opening of commitments as required by BL protocol), provides a proof that either the messages are honestly computed according to the randomness committed in the first round, or the "trapdoor" condition is satisfied. Formally, $\mathcal{P}$ provides rZAPs for the following **NP** languages (except the language Λ_{SHCom} that is proved only by $\mathcal{V}$ using

resSZK protocol).

Λ_{BL1}**:** correctness and consistency of the first round of Blum's protocol (BL1). A tuple $(x, m, \mathsf{c}_{r_b}, \mathsf{c}_b) \in \Lambda_{\mathsf{BL1}}$ if: there exist (r_b, s_b) such that $\mathsf{c}_{r_b} = \mathsf{SBCom}(r_b, s_b)$ and m is honestly computed according to BL1 for the graph x using randomness $f_{r_b}(\mathsf{c}_b)$.

Λ_{V}**:** correctness and consistency of verifier's messages of the protocol resSZK. A tuple $(m_P, m_V, \mathsf{c}_{r_b}, \mathsf{c}_b) \in \Lambda_{\mathsf{V}}$ if: there exist (r_b, s_b) such that $\mathsf{c}_{r_b} = \mathsf{SBCom}(r_b, s_b)$ and m_V is honestly computed according to the verifier's procedure of the protocol resSZK having in input prover's message m_P (m_P corresponds to the concatenation of all messages played by the prover so far) using randomness $f_{r_b}(\mathsf{c}_b)$.

Λ_{trap}**: trapdoor theorem** (true only for sub-protocol b). The pair $(\mathsf{c}_s, b) \in \Lambda_{\mathsf{trap}}$ if there exists s such that $\mathsf{c}_s = \mathsf{SBCom}(b, s)$.

Λ_{SHCom}**:** validity of the opening (proved by $\mathcal{V}$). The pair $(\mathsf{c}_s, m) \in \Lambda_{\mathsf{SHCom}}$ if there exists s such that $\mathsf{c}_s = \mathsf{SHCom}(m, s)$. Note that for a statistically hiding commitment scheme, any pair (c_s, m) is actually in Λ_{SHCom}. Nevertheless, $\mathcal{V}$ proves this theorem using the argument of knowledge resSZK.

Protocol sim$\mathsf{res}\mathcal{WI}\mathsf{AoK}$ consists of two phases (see Fig. 6.2). In the first phase, $\mathcal{P}$ and $\mathcal{V}$ generate the random tapes that they will use to run the sub-protocols. $\mathcal{P}$ sends $\mathcal{V}$ the commitments $\mathsf{c}_{r_0}, \mathsf{c}_{r_1}$ of two random strings r_0, r_1 and the commitment c_s of a random bit b. This message is the *determining message* on which $\mathcal{V}$ applies a PRF to generate a pseudo-random tape (to be used to execute the sub-protocols). The second phase consists of a parallel execution of π_0 and π_1 (see Fig. 6.3). $\mathcal{P}$ runs each sub-protocol on theorem x, randomness r_0, r_1, and the witnesses for computing the rZAPs as inputs (i.e., the opening of the commitments of the determining message). $\mathcal{V}$ runs each sub-protocol using the pseudo-random tapes determined by the determining message received from $\mathcal{P}$. Each sub-protocol is resettable $\mathcal{WI}$, while only one of the two sub-protocols is resettably-sound and a resettable AoK. Since $\mathcal{V}$ accepts the proof only if *both* executions are accepting, the final protocol is also a resettably-sound resettable AoK.

The sub-protocol Π_d is described in Fig. 6.3. We omit the first round of the rZAP and the first round of the statistically hiding commitment scheme SHCom. rZAPs are computed with independent randomness. We stress out that the determining message for $\mathcal{V}$ is the first prover's message: $\mathsf{dm} = (\mathsf{c}_{r_0}, \mathsf{c}_{r_1}, \mathsf{c}_s)$. The determining message for $\mathcal{P}$ is the first verifier's message: $(\mathsf{c}_0, \mathsf{c}_1)$.

6.2.2 Security Proof

In this section we provide the high-level proof of the simultaneous resettable witness indistinguishability property and the resettable argument of knowledge property of the protocol depicted in Fig. 6.2.

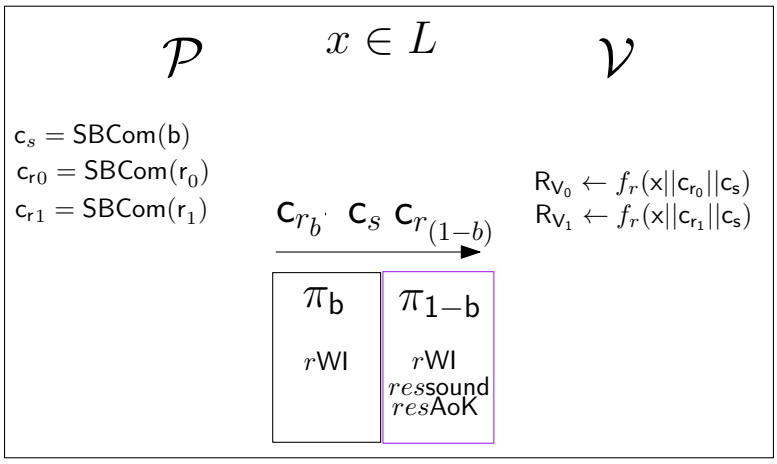

Figure 6.1: *Graphic Representation of* simres$\mathcal{WI}$AoK.

Protocol simres$\mathcal{WI}$AoK

Inputs: common input $x \in \mathsf{HC}$.
$\mathcal{P}$'s input: witness y, randomness ω. $\mathcal{V}$'s input: randomness r.

$\mathcal{P}$**:** $b \xleftarrow{\$} \{0,1\}$; $r_0, r_1, s_0, s_1 \xleftarrow{\$} \{0,1\}^n$.
Send $\mathsf{c}_{r_0} \leftarrow \mathsf{SBCom}(r_0, s_0)$, $\mathsf{c}_{r_1} \leftarrow \mathsf{SBCom}(r_1, s_1)$, $\mathsf{c}_s \leftarrow \mathsf{SBCom}(b, s)$.
Run in parallel $\Pi_0^{\mathcal{P}}(x, y, r_0, s_0)$; $\Pi_1^{\mathcal{P}}(x, y, r_1, s_1)$.

$\mathcal{V}$: upon receiving $\mathsf{dm} = (\mathsf{c}_{r_0}, \mathsf{c}_{r_1}, \mathsf{c}_s)$ from $\mathcal{P}$.
$\mathsf{R}_{\mathsf{V}0} \leftarrow f_r(x||\mathsf{c}_{r_0}||\mathsf{c}_s)$; $\mathsf{R}_{\mathsf{V}1} \leftarrow f_r(x||\mathsf{c}_{r_1}||\mathsf{c}_s)$;
Run in parallel $\Pi_0^{\mathcal{V}}(x, \mathsf{R}_{\mathsf{V}0})$; $\Pi_1^{\mathcal{V}}(x, \mathsf{R}_{\mathsf{V}1})$.

Figure 6.2: *Simultaneously Resettable Argument of Knowledge.*

Resettable-soundness. Towards showing resettable soundness we start with the following observations. Recall that by dm we denote the determining message sent by $\mathcal{P}^*$ in the first round consisting of the commitment of two random seeds and the commitment of a bit (let us call the bit committed b).

1. The randomness used by $\mathcal{V}$ depends on dm. In a resetting attack, malicious prover $\mathcal{P}^*$ activates $\mathcal{V}$ by selecting theorem and randomness, denoted by (x, j) which forces $\mathcal{V}$ to run with the same randomness r_j among several executions. However, the randomness actually used by $\mathcal{V}$ at each session is determined by the output of the PRF on seed r_j and input (x, dm). Thus, even if activated with the same random tape r_j, when receiving a new determining message, $\mathcal{V}$ executes the protocol with a fresh pseudo-random tape. Note that, due to the computational indistinguishability of the PRF, soundness holds against a computationally bounded adversary.

Sub-protocol: $\Pi_d = \langle \Pi_d^{\mathcal{P}}(x, y, r_d, s_d), \Pi_d^{\mathcal{V}}(x, \mathsf{R}_{\mathsf{V}d}) \rangle$.
Inputs: common input: x ($\in \mathsf{HC}$). $\mathcal{P}$'s input: witness y for $\mathcal{R}_{\mathsf{HC}}$; witness (r_d, s_d) to prove rZAP's consistency theorem. $\mathcal{V}$'s input: randomness $\mathsf{R}_{\mathsf{V}d}$. Protocols BL [12] and resSZK [83] are used as sub-protocols.

- $\mathcal{V}$: Pick challenge for BL protocol: $ch_d \xleftarrow{\$} \{0,1\}^n$. Send $\mathsf{c}_d \leftarrow \mathsf{SHCom}(ch_d)$ to $\mathcal{P}$.
- $\mathcal{P}$: upon receiving c_d (this is the determining message for $\mathcal{P}$):
  1. Generate randomness $\mathsf{R}_{\mathsf{P}d} \leftarrow f_{r_d}(x||\mathsf{c}_d)$.
  2. Compute the step BL1 for the instance x using randomness $\mathsf{R}_{\mathsf{P}d}$. Let us denote the output as $\mathsf{mBL1}^d$.
  3. Send $\mathsf{mBL1}^d$ to $\mathcal{V}$ along with the rZAP for theorem: $((x, \mathsf{mBL1}^d, \mathsf{c}_{r_d}, \mathsf{c}_d) \in \Lambda_{\mathsf{BL1}} \vee (\mathsf{c}_s, d) \in \Lambda_{\mathsf{trap}})$.
- $\mathcal{V}$: if rZAP is accepting send ch_d to $\mathcal{P}$.
  Prove theorem $(\mathsf{c}_d, ch_d) \in \Lambda_{\mathsf{SHCom}}$ using resSZK protocol. Let $m^d_{\mathsf{P}_{\mathsf{rszk}}}$ be the prover's message of sub-protocol resSZK (sent by $\mathcal{V}$ to $\mathcal{P}$) and $m^d_{\mathsf{V}_{\mathsf{rszk}}}$ be the verifier's message of resSZK (sent by $\mathcal{P}$ to $\mathcal{V}$):
  1. $(\mathcal{P} \to \mathcal{V})$ at each round of the protocol resSZK, upon receiving $m^d_{\mathsf{P}_{\mathsf{rszk}}}$ from $\mathcal{V}$, $\mathcal{P}$ computes $m^d_{\mathsf{V}_{\mathsf{rszk}}}$ using randomness $\mathsf{R}_{\mathsf{P}d}$ and sends $m^d_{\mathsf{V}_{\mathsf{rszk}}}$ to $\mathcal{V}$ along with an rZAP for the theorem $((m^d_{\mathsf{P}_{\mathsf{rszk}}}, m^d_{\mathsf{V}_{\mathsf{rszk}}}, \mathsf{c}_{r_d}, \mathsf{c}_d) \in \Lambda_{\mathsf{V}} \vee (\mathsf{c}_s, d) \in \Lambda_{\mathsf{trap}})$.
  2. $(\mathcal{V} \to \mathcal{P})$ at each round of the protocol resSZK upon receiving $m^d_{\mathsf{V}_{\mathsf{rszk}}}$ from $\mathcal{P}$, if rZAP is accepting $\mathcal{V}$ computes the next resSZK's prover message and sends it to $\mathcal{P}$. Otherwise it aborts.
- $\mathcal{P}$: upon successfully completing the resSZK protocol compute step BL3 and send the message $\mathsf{mBL3}^d$ to $\mathcal{V}$.
- If $\mathsf{mBL3}^d$ is the correct third message of BL protocol $\mathcal{V}$ outputs `accept`, else outputs `abort`.

Figure 6.3: *Sub-protocol* $\Pi_d = (\Pi_d^{\mathcal{P}}(\cdot), \Pi_d^{\mathcal{V}}(\cdot))$.

2. In sub-protocol Π_b, the resetting power of $\mathcal{P}^*$ is effective since $\mathcal{P}^*$ can honestly prove the trapdoor theorem of the rZAP. Therefore, $\mathcal{P}^*$ is not forced to use the randomness committed in the determining message among multiple resetting attacks. Specifically, $\mathcal{P}^*$ can mount the following attack. $\mathcal{P}^*$ initiates a session labelled by (x, j, dm). In the sub-protocol Π_b, upon the reception of challenge ch_b from $\mathcal{V}$, $\mathcal{P}^*$ resets $\mathcal{V}$ (while keeping the *same* determining message) back to the second round (the point after $\mathcal{V}$ has sent the commitment of the challenge). Then, $\mathcal{P}^*$ changes the message $\mathsf{mBL1}^b$ according to the challenge ch_b previously seen. This is possible using the trapdoor theorem, therefore $\mathcal{P}^*$ does not need to stick with the randomness committed in the determining message. Since the determining message is the same as before the reset, $\mathcal{V}$ will use the same challenge in the sub-protocol Π_b. Thus, in this sub-protocol, $\mathcal{P}^*$ can prove any theorem by

obtaining the challenge in advance and thus Π_b is not resettable sound.

3. In sub-protocol $\Pi_{\bar{b}}$, the trapdoor theorem is always false, thus resetting $\mathcal{V}$ is ineffective. Indeed, in order to provide an accepting transcript, $\mathcal{P}^*$ must provide an rZAP that only exists when the "consistency" theorem is true, that is, each of $\mathcal{P}^*$'s message is honestly computed according to the randomness committed in the determining message. By the statistically binding property of SBCom (there exists only one opening for the commitments c_s and $\mathsf{c}_{r_{\bar{b}}}$) and the soundness of rZAP (any unbounded $\mathcal{P}^*$ cannot prove a false theorem), $\mathcal{P}^*$ must be consistent with the randomness committed in the determining message. Therefore, $\Pi_{\bar{b}}$ is resettably sound.

Assume that there exists a PPT malicious prover $\mathcal{P}^*$ and a pair (x, j) such that $\mathcal{V}$ accepts x with non-negligible probability for some $x \notin \mathsf{HC}$. By observation 1, such a transcript is indexed by determining message dm. Thus, the accepting transcript can be labelled by triple (x, j, dm). By observation 2, for the same determining message dm, there are polynomially many distinct transcripts for sub-protocol Π_b ($\mathcal{P}^*$ can reset $\mathcal{V}$ polynomially many times and change the protocol messages). All these (partial) transcripts of Π_b can be accepting for $x \notin \mathsf{HC}$ since soundness does not hold for Π_b. However, by observation 3, for a fixed triple (x, r_j, dm), there exists only one possible accepting transcript for sub-protocol $\Pi_{\bar{b}}$ since $\mathcal{P}^*$ is forced to honestly follow the BL protocol according to the randomness committed in the determining message. Therefore the soundness of BL is preserved when $\mathcal{P}^*$ resets $\mathcal{V}$ in $\Pi_{\bar{b}}$. Since $\mathcal{V}$ accepts if and only if the executions of *both* sub-protocols are accepting, protocol $\mathsf{simres}\mathcal{WI}\mathsf{AoK}$ is resettably sound.

Resettable argument of knowledge. To prove resettable argument of knowledge we show an expected PPT extractor that extracts the witness from any malicious prover $\mathcal{P}^*$ with probability that is negligibly close to the probability that $\mathcal{P}^*$ convinces an honest verifier. Let (x, j, dm) be the label of the session in which $\mathcal{P}^*$ provides an accepting proof. The goal of the extractor is to obtain two accepting transcripts with the same $\mathsf{BL1}$ message and two distinct challenges (for at least one sub-protocol) for the same label.

Our extractor consists of two phases. In the first phase it follows the honest verifier procedure. When $\mathcal{P}^*$ has completed its execution, if there exists an accepting session labeled by (x, j, dm) that we call "target session", the extractor proceeds to the second phase. In the second phase, the extractor obtains a distinct accepting transcript for the target session by cheating in the "opening" of the commitment by sending a challenge that is distinct from the one sent in the first phase and simulating the zero knowledge proof given by the verifier.

The crucial step of this phase is to detect the sub-protocol in which $\mathcal{P}^*$ is stuck with the randomness committed in dm and must follow the protocol honestly. Indeed, in such sub-protocol, the extractor can use the stand-alone simulator and open the statistically hiding commitment to any challenge. Note that the non-black-box simulator of the protocol resSZK takes as input the code of the malicious verifier. Thus, in order to use

the simulator, the extractor must carefully prepare a machine which internally handles the interaction with $\mathcal{P}^*$ and forwards to the simulator only the messages belonging to the resSZK protocol played in one of the sub-protocol. One of the tasks of such machine is detecting the sub-protocol in which $\mathcal{P}$ is forced to be honest. Once the right sub-protocol has been detected, by the statistically-hiding property of SHCom, and by the statistical zero-knowledge property of protocol resSZK run by $\mathcal{V}$ instead of the opening, we are guaranteed that upon each rewind, $\mathcal{P}^*$ provides another accepting transcript for the target session with the same probability of the first phase. Finally, by the proof of knowledge property of Blum's protocol, collecting two distinct transcripts allows the extractor to compute the witness. The actual extractor requires an intermediate estimation step (as shown in [48]) in which the probability of having another accepting transcript for the label (x, j, dm) is estimated. More details on the formal description of the extractor, the augmented machine and the formal proof can be found in the full version of this work.

The actual extractor requires an intermediate estimation step (as shown in [48]) in which the extractor estimates the probability of having another accepting transcript for the label (x, j, dm).

Therefore, the random tape used by $\mathbf{E}$ can be seen as partitioned in three blocks. The first block is used to activate the malicious prover, the second block is used to executed the first phase as the honest verifier, the size of this block is a fixed polynomial $s(n)$ and depends of the malicious $\mathcal{P}^*$. We denote the first block by R^* and the second block as $\vec{R}$. The last block that we denote as R' is used to perform the estimation phase and the second phase. The size of the second block is an arbitrary polynomial.

The formal description of the extractor is shown in Figure 6.4. The augmented machine is described in Fig.6.5.

Remark 4. *The above extractor follows the behaviour of any standard extractor, therefore when it halts, it either outputs the witness or the special symbol $\perp$. However, if the* simres$\mathcal{WI}$AoK *protocol is used as sub-routine in a larger protocol, when proving the security of the larger protocol, it could be useful to have an extractor that in case of failure provides more information, that can be used by a possible outer simulator/extractor. Therefore it is straightforward to modify the above extractor such that in case of abort it outputs the last message received from $\mathcal{P}^*$.*

Claim 6.2.1. *The extractor* $\mathbf{E}$ *runs in expected polynomial time in the security parameter n.*

Proof. The extractor consists of three phases: the honest verifier phase, the estimation phase and the extraction phase. The honest verifier phase consists in executing the (PPT) procedure of the honest verifier, thus this step requires polynomial time $t_{\mathtt{ver}}(n)$. Now, assume that in this phase $\mathcal{P}^*$ has provided an accepting transcript for a session (x, j, dm) (as explained above, a session is determined also by the determining message dm), with probability $\zeta^{(x,j)} = \zeta(R^*, r_j, x)$. Then, with probability $\zeta^{(x,j)}$ $\mathbf{E}$ initiates to the estimation phase.

Resettable Extractor $\mathbf{E}(desc(\mathcal{P}^*), R^*, \vec{R}, R')$
Input: Random tape for honest phase $\vec{R} = r_1, \ldots, r_{s(n)}$, R^* for $\mathcal{P}^*$'s activation, for all other computations R'.

- **Honest Verifier Phase.** Run $\langle \mathcal{P}^*(R^*), \mathcal{V}(\vec{R}) \rangle$. Upon completion of $\mathcal{P}^*$'s execution: let (x, j, dm) the label of the last accepting session, and let $\tau_{x,j}$ be the accepting transcript (for better readability we omit the subscript dm). If there are no accepting transcripts output $\perp$. From now on, the extractor will focus on obtaining another accepting transcript $\tau'_{x,j}$ for the same session that we call target session. For all the other sessions, the extractor proceeds as the honest verifier. Fixed the label (x, j, dm) the accepting transcript can be seen as the concatenation of the accepting transcript of each sub-protocol $\tau_{x,j} = \tau^0_{x,j}, \tau^1_{x,j}$ and dm.
- **Estimation Phase.** Run $\mathcal{P}^*(R^*)$ and execute the honest verifier procedure using randomness $\vec{R}$ for all sessions except that in the target session execute the protocol resSZK with independent fresh randomness taken from R'. Repeat this step until either n^2 accepting transcripts for the target session have been obtained, or the loop has been repeated 2^n times. Let $q(n)$ be the number of iteration needed to obtain the n^2 accepting transcripts.
- **Extraction Phase.** Pick $b \xleftarrow{\$} \{0,1\}$.
  Repeat $q(n)$ times:
  1. pick $ch' \leftarrow \{0,1\}^n$ using random tape R'. Run $\tau_{ts} \leftarrow \mathsf{Sim}(\mathbf{M}(\mathcal{P}^*, \mathcal{V}, R^*, \vec{R}, b, (x, j, \mathsf{dm}), \tau^b_{x,j}, ch'))$ on the theorem $(\mathsf{c}_b, ch') \in \Lambda_{\mathsf{SHCom}}$; If Sim does not abort, **reconstruct the new accepting transcript:** run $\mathcal{P}^*(R^*)$ and plays as the honest verifier with randomness $\vec{R}$ except that in all sessions labelled with (x, j, dm) run the sub-protocol $\Pi^{\mathcal{V}}_b$ opening the commitment c_b as ch' and using the transcript τ_{ts} for the $\mathsf{rs}S\mathcal{Z}\mathcal{K}$ proof. At the end of the execution obtain the transcript $\tau_{x,j} = \tau'^b_{x,j}, \tau^{(\bar{b})}_{x,j}$ and use $\tau^b_{x,j}$ and $\tau'^b_{x,j}$ to extract the cycle y. Output y and halt.
  2. $b \leftarrow (b+1) mod\ 2$.

  Output $\perp$.

Figure 6.4: *The resettable extractor* $\mathbf{E}$.

The estimation phase, follows the Goldreich Kahan [48] technique, and consists of repeating the execution with $\mathcal{P}^*$ until n^2 accepting transcript for the session (x, j, dm) are obtained. Upon each repetition, the view of $\mathcal{P}^*$ is identical for all the other sessions, and for session (x, j, dm) the only change is the randomness used by the verifier in running the protocol resSZK. Therefore, we have that at each repetition, in session (x, j, dm), $\mathcal{P}^*$ produces an accepting transcript with probability $\zeta^{(x,j)}$. Therefore in order to obtain n^2 accepting transcript $\mathbf{E}$ runs the second step: $q(n) = \frac{n^2}{\zeta^{(x,j)}}$ times.
In the extraction phase $\mathbf{E}$ runs the simulator Sim on input the augmented machine $\mathbf{M}$. The augmented machine takes in input (among the other inputs) the bit of the target sub-protocol in which the extractor wants to cheat and does the following: 1) it executes the honest verifier procedure as long as it does not detect that the prover is successfully

resetting the verifier in the target sub-protocol 2) forwards the messages belonging to the resSZK protocol of the target sub-protocol to Sim. Hence, **M** runs in polynomial time. The non-black box simulator of the protocol resSZK runs also in polynomial time $t_{\mathtt{sim}}$. Recall that the extractor has to figure out in which sub-protocol it can safely use the stand-alone simulator. In order to do this, at each attempt **E** invokes Sim first with input **M** that cheats in sub-protocol b and the then again **M** cheating in sub-protocol $(1-b)$.

Thus, the extraction phase consists of the repetition of at most $2q(n)$ times of the simulator Sim plus a polynomial time due to the reconstruction of the new transcript that we denote as $t_{\mathtt{rec}}$.

Summing up the total running time of the three phases is the following:

$$t_{\mathtt{ver}} + \zeta^{(x,j)} \cdot \Big[q(n) \cdot \mathtt{poly} + t_{\mathtt{sim}} \cdot q(n) + t_{\mathtt{rec}}\Big] = \mathtt{poly}(n)$$

□

Claim 6.2.2. *Let $\mathcal{P}^*$ be a PPT prover and let p the probability that $\mathcal{P}^*$ provides an accepting transcript for a theorem $x \in \mathsf{HC}$. If trapdoor permutations and collision-resistant hash functions exist, then* **E** *outputs the witness $y \in \mathcal{R}_{\mathsf{HC}}(x)$ with probability only negligibly far from p.*

Proof. Recall that due to the statistical biding of SBCom and to the soundness of rZAPs, for each accepting session (x, j, dm) the sub-protocol $\Pi_{\bar{b}}$ is resettably sound and that in such sub-protocol $\mathcal{P}^*$ is stuck with the randomness committed in the determining message dm. Note that the strategy of the extractor is basically to play honestly in all sessions, while in session (x, j, dm) it tries to rewind by keeping the same verifier's determining message, i.e. the commitments of the challenges of BL's protocol $\mathsf{c}_0, \mathsf{c}_1$ and cheating in the opening by providing a false proof. Such a cheating is allowed only in the sub-protocol $\Pi_{\bar{b}}$ where $\mathcal{P}^*$ is forced to be consistent and in turn for **E** is sufficient to cheat invoking the only stand-alone non-black box simulator Sim. Assume that there exists a session (x, j, dm) (that we will denote as target session) in which $\mathcal{P}^*$ generates an accepting transcript with probability p. An accepting transcript consists of a pair of sub-transcripts $\tau_{x,j} = (\tau^0_{x,j}, \tau^1_{x,j})$. We want to argue that **E** is able to obtain from $\mathcal{P}^*$ a new accepting sub-transcript $\tau^{\prime(\bar{b})}_{x,j}$, for the target session with almost the same probability. We show this through hybrids arguments.

H_0 : In this hybrid consider a modified version of the extractor **E** that runs always as the honest verifier. Namely, in the third phase (the extraction phase), instead of playing with challenge ch', **E** activated the augmented machine **M** on input the honest openings of the challenge ch (as the honest receiver) and the messages of the augmented machine are held by the real prover instead of the simulator. Obviously in this modified extraction phase, once **E** gets another accepting transcript for the target session it does not extract the witness. In this experiment the view of the prover $\mathcal{P}^*$ interacting with an honest $\mathcal{V}$ is indistinguishable from the view of $\mathcal{P}^*$

interacting with the modified $\mathbf{E}$. The only difference is that here the extractor could abort more often than honest $\mathcal{V}$.

Now we want to argue that in this experiment the extractor outputs $\perp$ with negligible probability. Note that the extractor outputs $\perp$ in the third phase if, after repeating the execution with $\mathcal{P}^*$ $2q(n)$ times (where $\frac{1}{q(n)}$ is an estimation of p), $\mathcal{P}^*$ does not provide another accepting transcript for the target session (x, j, dm). Note that by the Goldreich-Kahan analysis we have that after $q(n)$ number of repetitions, $\mathcal{P}^*$ provides another accepting transcript with overwhelming probability. Note also that, in the extraction phase, the extractor runs the augmented machine $\mathbf{M}$. More specifically each repetition $\mathbf{M}$ is run with a target sub-protocol in input, and it aborts when detects that the prover's messages are not consistent with previous transcript. However, at each repetition $\mathbf{M}$ is run twice, once per each sub-protocol. Now, by the soundness of rZAP and by the statistically-binding property of the commitment sent by $\mathcal{P}^*$, we have that at least in one sub-protocol $\mathcal{P}^*$ must be consistent with previous transcript. Then, since in the extraction phase, each attempt is repeated once for each sub-protocol, there exists a sub-protocol in which $\mathbf{M}$ does not abort. Therefore, at each repetition, $\mathbf{E}$ aborts only if also $\mathcal{P}^*$ aborts. Due to the indistinguishability of the view generated by $\mathbf{E}$, $\mathcal{P}^*$ aborts with the same probability of the first and second phase. Thus the probability of obtaining a new transcript in the third phase is overwhelming, and in turn the probability that $\mathbf{E}$ outputs $\perp$ is negligible.

H_1: In this hybrids the extractor works as in experiment H_1 except that in the third phase instead of handling the message of the augmented machine $\mathbf{M}$ running the honest prover strategy on the theorem $(\mathsf{c}_b, ch \in \Lambda_{\mathsf{V}})$ it invokes the zero knowledge simulator Sim. By the statistically zero knowledge property of the protocol resSZK H_0 and H_1 are statistically close.

H_2: In this hybrid the extractor works as in the previous hybrid except that in the third phase it invokes the simulator on theorem $(\mathsf{c}_b, ch' \in \Lambda_{\mathsf{V}})$. Due to the statistically hiding property of the commitment SHCom hybrids H_2 and H_1 are statistically close. This is the extractor described in Figure 6.4.

Remark 5 (Simulation Soundness in not needed.)**.** *Note that in this experiment $\mathcal{P}^*$ is receiving simulated proofs by* Sim *of a false theorem and could maul this proof in other concurrent executions. Since* resSZK *is not simulation sound (very roughly a protocol is simulation sound when even in case the adversary receives simulated proofs of false theorems it is still not able to prove a false theorem to an honest verifier) we cannot rule out this possibility. Note however that by the unconditional soundness of* rZAP*, $\mathcal{P}^*$ cannot use the simulated proof within the same execution in which the simulator is used. Still $\mathcal{P}^*$ could open new concurrent executions and use the simulated messages to lead $\mathcal{V}$ to accept a false theorem. This is not a problem since the extractor, once is in the third phase, has already a target theorem/session on which is trying to extract, therefore, other new (possibly false) theorems proved*

in other sessions are nor relevant to $\mathbf{E}$.

□

Augmented machine. In this paragraph we formally define the augmented machine $\mathbf{M}$ (depicted in Figure 6.5). Very roughly, $\mathbf{M}$ internally runs the system $\langle \mathcal{P}^*(R^*), \mathcal{V}(\vec{R}) \rangle$ honestly using the same randomness used in the first phase of $\mathbf{E}$ for all sessions different from the target session. The target session (x, j, dm), and the sub-protocol Π_b in which $\mathbf{M}$ has to cheat are provided in input. The cheating consists in opening the commitment of the challenge c_b sent in the first phase as a fresh challenge ch' that is also provided as input (note that $\mathbf{M}$ is a deterministic machine).

For the target session, $\mathbf{M}$ cheats by simulating the ZK protocol proving that ch' is the correct opening of c_b, i.e., the verifier's message of resSZK sent by $\mathcal{P}^*$ are written to the output tape, and $\mathbf{M}$ waits that the simulator writes the prover's answer to the input tape. One of the most important tasks of $\mathbf{M}$ is to detect if the sub-protocol Π_b in which it is trying to cheat is the wrong one, i.e. is the one in which $\mathcal{P}^*$ is free to reset the verifier without getting caught. In order to do this, $\mathbf{M}$ will receive in input the transcript of the target sub-protocol $\tau^b_{x,j}$ generated in the first phase, such that it can check if the messages sent by $\mathcal{P}^*$ are consistent with such transcript and thus detect if $\mathcal{P}^*$ is changing her messages among the resets.

Summing up, $\mathbf{M}$ receives the following inputs:

- the code of $\mathcal{P}^*$ and $\mathcal{V}$, the randomness $R^*, \vec{R}$, and the target session (x, j, dm) to reproduce the same execution generated in the first phase by $\mathbf{E}$ for all sessions except the target session (x, j, dm);
- the bit b indicates the sub-protocol Π_b of the target session in which the messages belonging to the resSZK protocol must be forwarded in output to the simulator;
- the fresh challenge ch' to be open to, that replaces the honest challenge committed in c_d;
- the transcript $\tau^b_{x,j}$ of the sub-protocol Π_b obtained in the first step of $\mathbf{E}$ and that $\mathbf{M}$ uses to detect if the sub-protocol Π_b is the one in which $\mathcal{P}^*$ is free to cheat.

In the following, we denote by $(a_1, \ldots, a_n) \neq (a'_1, \ldots, a'_n)$ the fact that there exists $i \in [n]$ such that $a_i \neq a'_i$. We denote as $\emptyset$ the empty string. When writing the Augmented Machine $\mathbf{M}$ we can not consider sub-protocols Π_0, Π_1 as black boxes, but we have to deal with each sub-protocol round. Following there is some notation for that. We indicate with $\mathsf{ZAP}^0_{\mathsf{BL}}$, $\mathsf{ZAP}^1_{\mathsf{BL}}$ the rZAPs sent along with messages $\mathsf{mBL1}^0, \mathsf{mBL1}^1$, and with $\mathsf{ZAP}^0_{\mathsf{rszk}}, \mathsf{ZAP}^1_{\mathsf{rszk}}$ the rZAPs sent along with each message $m^0_{\mathsf{V}_{\mathsf{rszk}}}, m^1_{\mathsf{V}_{\mathsf{rszk}}}$ of the resSZK protocol. We denote as $\tau^b_{x,j}$ the transcript of the sub-protocol b (the sub-session from which we are trying to extract the witness) for the accepting session labelled with (x, j, dm). Recall that $\tau^b_{x,j}$ was generated by the extractor in the first phase (honest verifier phase)(see Figure 6.4). A sub-protocol Π_d consists of three stages, the $\mathsf{BL1}$ step (along with ZAPs), the resSZK protocol and the $\mathsf{BL3}$ phase. In particular $\mathsf{BL1}, \mathsf{BL3}$ consist of a single message from $\mathcal{P}$ to $\mathcal{V}$. The resSZK steps consists of ℓ messages. Thus we denote with $\tau^b_{x,j}[\mathsf{BL1}], \tau^b_{x,j}[\mathsf{BL3}]$ the single messages for $\mathsf{BL1}, \mathsf{BL3}$ steps along with re-

spective rZAP, and with $\tau^b_{x,j}[\mathsf{resSZK}^i]$ the i-th verifier's message of protocol resSZK. All messages are considered along with the respective rZAPs. $\mathbf{M}$ stores in the local variable τ_{Sim} the transcript of the messages received by the simulator of resSZK protocol.

$\mathbf{M}(\mathcal{P}^*, \mathcal{V}, R^*, b, (x, j, \mathsf{dm}, \tau^b_{x,j}), \vec{R}, ch')$

$\tau_{\mathsf{Sim}} \leftarrow \emptyset$;
Run $\mathcal{P}^*(R^*)$. When receiving a command $(x', j', \mathsf{dm}', m_{\mathcal{P}^*})$:

- if $(x', j', \mathsf{dm}') \neq (x, j, \mathsf{dm})$: return $\mathcal{V}(x', r_j, \mathsf{dm}', m_{\mathcal{P}^*})$;
- else:
  - if $m_{\mathcal{P}^*} = \emptyset$:
    - $(\mathsf{R}_{\mathsf{V}0}|\mathsf{R}_{\mathsf{V}1}) \leftarrow f_{r_j}(x||\mathsf{dm})$;
    - $\mathsf{c}_b \leftarrow \Pi^{\mathcal{V}}_b(x, \mathsf{R}_{\mathsf{V}b}, \mathsf{mBL1}^b, \mathsf{ZAP}^b_{\mathsf{BL}})$; $\mathsf{c}_{\bar{b}} \leftarrow \Pi^{\mathcal{V}}_{\bar{b}}(x, \mathsf{R}_{\mathsf{V}\bar{b}}, \mathsf{mBL1}^{\bar{b}}, \mathsf{ZAP}^{\bar{b}}_{\mathsf{BL}})$. Return $\mathsf{c}_0, \mathsf{c}_1$ to $\mathcal{P}^*$;
  - if $m_{\mathcal{P}^*} = (\mathsf{mBL1}^0, \mathsf{ZAP}^0_{\mathsf{BL}}, \mathsf{mBL1}^1, \mathsf{ZAP}^1_{\mathsf{BL}})$:
    - if $(\mathsf{mBL1}^b, \mathsf{ZAP}^b_{\mathsf{BL}}) \neq \tau^b_{x,j}[\mathsf{BL}]$ `ABORT`;
    - else compute $ch_{\bar{b}} \leftarrow \Pi^{\mathcal{V}}_{\bar{b}}(x, \mathsf{R}_{\mathsf{V}\bar{b}}, \mathsf{mBL1}^{\bar{b}}, \mathsf{ZAP}^{\bar{b}}_{\mathsf{BL}})$. Return $ch', ch_{\bar{b}}$ to $\mathcal{P}^*$.
  - if $m_{\mathcal{P}^*} = (\mathsf{mBL1}^0, \mathsf{ZAP}^0_{\mathsf{BL}}, \mathsf{mBL1}^1, \mathsf{ZAP}^1_{\mathsf{BL}}, m^{0,i}_{\mathsf{V}_{\mathsf{rszk}}}, \mathsf{ZAP}^0_{\mathsf{rszk}}, m^{1,i}_{\mathsf{V}_{\mathsf{rszk}}}, \mathsf{ZAP}^1_{\mathsf{rszk}})$ for $i \in [\ell]$:
    - if $(\mathsf{mBL1}^b, \mathsf{ZAP}^b_{\mathsf{BL}}) \neq \tau^b_{x,j}[\mathsf{BL}]$ or $(m^{b,i}_{\mathsf{V}_{\mathsf{rszk}}}, \mathsf{ZAP}^b_{\mathsf{rszk}}) \neq \tau^b_{x,j}[\mathsf{resSZK}^i]$ `ABORT`;
    - else $m^{\bar{b}}_{\mathsf{P}_{\mathsf{rszk}}} \leftarrow \Pi^{\mathcal{V}}_{\bar{b}}(x, \mathsf{R}_{\mathsf{V}\bar{b}}, \mathsf{mBL1}^{\bar{b}}, \mathsf{ZAP}^{\bar{b}}_{\mathsf{BL}}, m^{\bar{b},i}_{\mathsf{V}_{\mathsf{rszk}}}, \mathsf{ZAP}^{\bar{b}}_{\mathsf{rszk}}$; if there exists a pair $(m^{\bar{b},i}_{\mathsf{V}_{\mathsf{rszk}}}, m^i_{\mathsf{Sim}}) \in \tau_{\mathsf{Sim}}$, return $m^{\bar{b}}_{\mathsf{P}_{\mathsf{rszk}}}, m^i_{\mathsf{Sim}}$ to $\mathcal{P}$; else write on the output tape $m^{\bar{b},i}_{\mathsf{V}_{\mathsf{rszk}}}$ and wait for the message of Sim.

Figure 6.5: *Augmented Machine*

Resettable witness indistinguishability. Recall that the protocol mainly consists of a single message from $\mathcal{P}$ to $\mathcal{V}$, the determining message $(\mathsf{c}_{r_0}, \mathsf{c}_{r_1}, \mathsf{c}_s)$, and the parallel execution of Π_0 and Π_1. Such protocol can be seen as a parallel repetition of (Π_0, Π_1) where Π_b is the protocol Π_b augmented with the message $(\mathsf{c}_s, \mathsf{c}_{r_b})$ sent from $\mathcal{P}$ to $\mathcal{V}$ and $b = 0, 1$.

Assume that there exists a resetting PPT distinguisher $\mathcal{V}^*$ for (Π_0, Π_1). That is, $\mathcal{V}^*$ distinguishes whether $\mathcal{P}$ runs *both* protocols using witnesses sampled from distribution $Y_0 = \{\bar{y}^0(\bar{x})\}_{\bar{x}}$ or from distribution $Y_1 = \{\bar{y}^1(\bar{x})\}_{\bar{x}}$. Let us denote by $H_{0,0}$ the experiment in which $\mathcal{P}$ uses witnesses sampled from Y_0 when running both protocols $(\Pi_b, \Pi_{\bar{b}})$, where b is the bit committed in c_s, and by $H_{1,1}$ the experiment in which $\mathcal{P}$ uses witnesses sampled from Y_1 in both $(\Pi_b, \Pi_{\bar{b}})$. We prove by hybrid arguments that experiments $H_{0,0}$ and $H_{1,1}$ are computationally indistinguishable. Let n denote the number of theorems and t the bound on the prover's random tapes. Consider the following hybrids.

$H_{1,0}$**:** In this hybrid, in each session, $\mathcal{P}$ uses witnesses sampled from Y_1 to run protocol Π_b and the bit b is committed in the determining message in such session. The only difference between experiment $H_{1,0}$ and $H_{0,0}$ is in the witness used in Π_b. Assume that there exists a distinguisher between hybrids $H_{0,0}$ and $H_{1,0}$ then it is possible to construct an adversary $\mathcal{V}^*_{\mathsf{BL}}$ for the $\mathcal{WI}$ property of sub-protocol BL of Π_b. Note that, when b is the bit committed in the determining message, the trapdoor theorem is true in Π_b. $\mathcal{V}^*_{\mathsf{BL}}$, on input $(\bar{x}, Y_0, Y_1)$, runs $\mathcal{V}^*$ as sub-routine and honestly executes the protocol $\Pi_{\bar{b}}$ using the witness belonging to Y_0. Instead for the execution of Π_b it forwards the messages received from $\mathcal{V}^*$ and belonging to BL protocol to the external prover, while it simulates the remaining messages belonging to Π_b. The first difficulty in such reduction seems to be the fact that $\mathcal{V}^*$ can mount a reset attack asking the prover of Π_b to run with the same randomness while changing the challenge of BL protocol. Instead, $\mathcal{V}^*_{\mathsf{BL}}$ can only mount a concurrent attack against the external BL's prover. Nevertheless, $\mathcal{V}^*_{\mathsf{BL}}$ can replicate the same attack of $\mathcal{V}^*$ for the following reasons. The randomness of the honest prover executing protocol Π_b is computed on the determining message (the commitment of BL's challenge) received from $\mathcal{V}^*$. Due to the pseudo-randomness of PRF, when $\mathcal{V}^*$ changes the determining message the prover of Π_b plays with fresh randomness. By the resettably-sound argument of knowledge property of the resSZK protocol and by the computational binding property of SHCom we have that $\mathcal{V}^*$ can not maintain the same determining message and query the prover with two distinct BL's challenges. Thus the resetting power is suppressed and $\mathcal{V}^*_{\mathsf{BL}}$ can replicate the same attack as $\mathcal{V}^*$. The second difficulty is that for each protocol message the honest prover of Π_b is required to send a rZAP proving that the messages are consistent with the randomness committed in the determining message. However, in the reduction $\mathcal{V}^*_{\mathsf{BL}}$ forwards the messages received by an external prover of BL's protocol, therefore it can not prove the consistency with the determine message. Nevertheless, since we are in the case in which the trapdoor theorem is true, $\mathcal{V}^*_{\mathsf{BL}}$ can forward the external messages and computes the rZAPs using the witness of the trapdoor theorem. Due to the resettable $\mathcal{WI}$ property of rZAP such deviation from the honest prover is not detected by any PPT $\mathcal{V}^*$. Then, by the $\mathcal{WI}$ of BL protocol hybrids $H_{0,0}$ and $H_{1,0}$ are computationally indistinguishable.

$H^{i,j}_{0,1}$ (with $1 \leq i \leq n$, $1 \leq j \leq t$): In hybrid $H^{i,j}_{0,1}$, in session (i,j), $\mathcal{P}$ runs protocol $\Pi_{\bar{b}}$ using the witness sampled from Y_1, while protocol Π_b is run by using a witness sampled from Y_0, and b is the bit committed in the determining message of such session. The only difference between experiment $H^{i,j}_{0,1}$ and $H^{i-1,j-1}_{0,1}$ is that in experiment $H^{i,j}_{0,1}$, in session (i,j), the witness is sampled from Y_1 in the sub-protocol where the trapdoor theorem is false. Note that $H^{0,0}_{0,1} = H_{1,0}$. Assume that there exists a distinguisher between $H^{i,j}_{0,1}$ and $H^{i-1,j-1}_{0,1}$ then it is possible to construct an adversary for the hiding of the commitment scheme SBCom. The reduction works as follows. $\mathcal{A}$ playing in the hiding experiment obtains the challenge commitment C. Then it runs $\mathcal{V}^*$ as sub-routine and simulates the honest prover $\mathcal{P}$

as in experiment $H_{0,1}^{i-1,j-1}$, except that in session (i,j) it proceeds as follows. It computes $\mathsf{c}_{r_0}, \mathsf{c}_{r_1}$ as the honest prover, while it sets $\mathsf{c}_s = C$, and sends the first round to $\mathcal{V}^*$. Then $\mathcal{A}$ uniformly chooses a bit b and executes the protocol Π_b using a witness sampled from distribution Y_1 and protocol $\Pi_{\bar{b}}$ using the witness sampled from distribution Y_0. Note that $\mathcal{A}$ can run both sub-protocols without knowing the opening of C since also the honest $\mathcal{P}$ never uses such witness in the protocol execution. When $\mathcal{V}^*$ terminates its execution, $\mathcal{A}$ hands the output of $\mathcal{V}^*$ to the distinguisher and outputs whatever the distinguisher outputs. If C is a commitment of b then the experiment simulated by $\mathcal{A}$ is distributed identically to experiment $H_{0,1}^{i-1,j-1}$. Else if C is a commitment of $\bar{b}$ then the experiment is distributed as experiment $H_{0,1}^{i,j}$. By the computational hiding of SBCom we have that experiments $H_{0,1}^{i,j}$ and $H_{0,1}^{i-1,j-1}$ are computational indistinguishable.

$H_{1,1}$**:** In this hybrid, $\mathcal{P}$ uses a witness sampled from Y_1 to run protocol Π_b and the bit b is committed in the determining message. The only difference between experiment $H_{0,1}^{n,t}$ and experiment $H_{1,1}$ is in the witness used to run sub-protocol Π_b. By the same arguments put forth in proving the indistinguishability of hybrid $H_{1,0}$ and $H_{0,0}$, experiments $H_{0,1}^{n,t}$ and $H_{1,1}$ are computational indistinguishable. This completes the proof.

Theorem 13. *If trapdoor permutations and collision-resistant hash functions exist, then the protocol shown in Fig. 6.2 is a Simultaneously Resettable Witness Indistinguishable Argument of Knowledge.*

6.3 Simultaneously Resettable Identification Schemes

In this section, we present the second application of our main protocol, the first construction of a simultaneously resettable identification scheme. Identification schemes represent one of the most successful practical applications of cryptographic protocols. The basic goal of an identification scheme is to prevent an adversary $\mathcal{A}$ from impersonating a honest user $\mathcal{P}$ to another honest user $\mathcal{V}$. However, this is not sufficient for some applications. Indeed, consider the case in which $\mathcal{V}$ provides a service to $\mathcal{P}$, and the service is restricted only to a small community controlled by $\mathcal{V}$. Then, $\mathcal{P}$ could give to another party T that is not in the small community, some partial information about his secret that is sufficient for T to obtain the service from $\mathcal{V}$, while still T does not know $\mathcal{P}$'s secret. The proof of knowledge property allows us to do secure identification as well as preventing the attack described above. When the identification protocol is a proof of knowledge, the sole fact that T convinces $\mathcal{V}$ is sufficient to claim that one can extract the whole secret from T. This implies that T obtained $\mathcal{P}$'s secret key corresponding to his identity, and this is unlikely to happen in scenarios where the same secret key is used for other critical tasks such as digital signatures. As discussed in the introduction, our simultaneously resettable identification scheme follows the above proof of knowledge paradigm. This extends the previous work of Bellare et al. [9] to a setting in which every party can be reset. We emphasize that our simultaneously resettable identification

scheme is easily obtained from our main protocol simres$\mathcal{WI}$AoK, so achieving a constant round complexity.

Identification protocols secure against reset attacks. We introduce the notion of Reset-Reset-1 security as a generalization of the Concurrent-Reset-1 CR1 notion introduced in [9]. CR1 considers an adversary I, called impersonator, that plays in two phases. In the first phase, it interacts with a prover as a resetting verifier (Reset phase). In the second phase, it has no access to the prover anymore, but it tries to impersonate such a prover to an honest verifier (Concurrent phase). In the second phase, I is not allowed to reset the verifier. In our new definition Reset-Reset-1 (RR1) the impersonator is allowed to reset in both phases. The formal definition is a straightforward extension of the one given in [9] and can be found in the full version of this work.

The protocol $\mathcal{ID}$. The protocol is depicted in Fig. 6.6. Let $f : \{0,1\}^n \to \{0,1\}^*$ be a one-way function, let n be the security parameter. The public key of $\mathcal{P}$ is the pair $(\mathsf{pk}_0, \mathsf{pk}_1)$, the secret key is x_d for a randomly chosen bit d, such that $\mathsf{pk}_0 = f(x_d) \vee \mathsf{pk}_1 = f(x_d)$. The protocol simply consists in $\mathcal{P}$ running the simres$\mathcal{WI}$AoK protocol with $\mathcal{V}$ to prove that it *knows* the preimage of either pk_0 or pk_1. Formally, let $\Lambda_{\mathcal{ID}}$ be the following language $\Lambda_{\mathcal{ID}} = \{(y_0, y_1)$: there exists $x \in \{0,1\}^n$ s.t. $y_0 = f(x) \vee y_1 = f(x)\}$, then the identification scheme consists of $\mathcal{P}$ proving the statement $(\mathsf{pk}_0, \mathsf{pk}_1) \in \Lambda_{\mathcal{ID}}$ using simres$\mathcal{WI}$AoK.

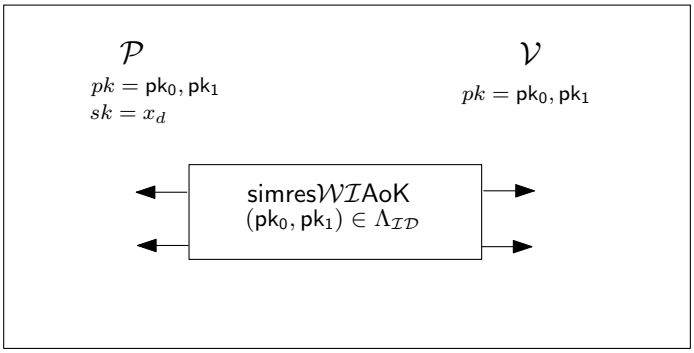

Figure 6.6: *Simultaneously Resettable ID scheme.*

Theorem 14. *If a constant-round simultaneously resettable $\mathcal{WI}$*AoK *protocol exists and one-way functions exist, then the above protocol is constant-round and secure in the* RR1 *setting.*

Proof. Let $pk = (\mathsf{pk}_0, \mathsf{pk}_1)$ be the public key of a player $\mathcal{P}$. Assume that there exists a PPT adversary I playing the RR1 experiment, that succeeds in impersonating an honest $\mathcal{P}$ with non-negligible probability. This means that I is able to prove to an honest $\mathcal{V}$ that her identity is $pk = (\mathsf{pk}_0, \mathsf{pk}_1)$. Then we show that I can be used to construct an adversary against the one-wayness of f, or a distinguisher for the resettable $\mathcal{WI}$

property of the simres$\mathcal{WI}$AoK protocol. The resettable argument of knowledge property of simres$\mathcal{WI}$AoK protocol is crucial to put forth both reductions.

Recall that, in the RR1 game, I plays the first phase interacting as a resetting verifier $\mathcal{V}^*$ with $\mathcal{P}$ and in the second phase interacts as resetting prover $\mathcal{P}^*$ with $\mathcal{V}$ trying to impersonate $\mathcal{P}$.

First we show an adversary $\mathcal{A}$ that breaks the one-wayness of f. $\mathcal{A}$ has in input a challenge y that is the output of $f(x)$ for some unknown x. The reduction works as follows. $\mathcal{A}$ picks $d \in \{0,1\}$, $x_d \in \{0,1\}^n$ and computes $\mathsf{pk}_d = f(x_d)$ and $\mathsf{pk}_{\bar{d}} = y$. Then it runs I as subroutine, in the first phase $\mathcal{A}$ simulates the honest prover playing the simres$\mathcal{WI}$AoK protocol with witness x_d. In the second phase, $\mathcal{A}$ simulates the honest verifier to I. If I provides an accepting proof, then $\mathcal{A}$ runs the extractor of the simres$\mathcal{WI}$AoK protocol and, by the resettable argument of knowledge property, except with negligible probability, it obtains the witness used by I in the proof. In order to run the extractor, $\mathcal{A}$ prepares an augmented machine that internally contains all messages belonging to the first phase so that they can be internally played with I, while the messages sent by I in the second phase are forwarded to the extractor. Now note that during the extraction process the extractor rewinds the machine several times changing the protocol messages (of the second phase), therefore I could change her messages accordingly. Note that however, since there is a separation between the first phase and the second phase, this does not require to re-play messages of the first phase. Since, by assumption f is a one-way function, the probability that the witness extracted corresponds to a pre-image of y is negligible.

Now, assume that the witness extracted from I is x_d. Then we can construct a distinguisher $\mathcal{A}_{\mathcal{WI}}$ for the resettable witness indistinguishability property of simres$\mathcal{WI}$AoK. $\mathcal{A}_{\mathcal{WI}}$ works as follows. It computes $\mathsf{pk}_0 = f(x_0), \mathsf{pk}_1 = f(x_1)$ and activates an external prover for the simres$\mathcal{WI}$AoK protocol with inputs $((\mathsf{pk}_0, \mathsf{pk}_1), (x_0, x_1))$. In the first phase, when I runs as a verifier, $\mathcal{A}_{\mathcal{WI}}$ forwards all messages to the external prover of the simres$\mathcal{WI}$AoK. In the second phase, when I runs as a prover, $\mathcal{A}_{\mathcal{WI}}$ follows the procedure of the honest verifier. Then, if I provides an accepting proof, then $\mathcal{A}_{\mathcal{WI}}$ runs the extractor of the simres$\mathcal{WI}$AoK protocol. Finally by the resettable argument of knowledge property, except with negligible probability, it obtains the witness used by I in the proof, i.e. it obtains x_0 or x_1. Now notice that in the previous experiment, when we tried to invert the one-way function, the witness extracted corresponded to the one used in the first phase, while I was verifying the proof. Since this second experiment is identical to the previous one, it is again true that the extracted witness corresponds to the one used by the prover. Since the prover now is the external prover of simres$\mathcal{WI}$AoK, we have that the above adversary $\mathcal{A}_{\mathcal{WI}}$ breaks the r$\mathcal{WI}$ property of simres$\mathcal{WI}$AoK. By the r$\mathcal{WI}$ property of simres$\mathcal{WI}$AoK, this event happens with negligible probability only and thus I wins the RR1 game with negligible probability. □

Bibliography

[1] Joël Alwen, Giuseppe Persiano, and Ivan Visconti. Impossibility and feasibility results for zero knowledge with public keys. In *Advances in Cryptology – Crypto '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 135–151. Springer Verlag, 2005. 122, 125, 163

[2] Seiko Arita. A constant-round resettably-sound resettable zero-knowledge argument in the bpk model. Cryptology ePrint Archive, Report 2011/404, 2011. http://eprint.iacr.org/. 164

[3] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Fran**c**cois-Xavier Standaert, and Christian Wachsmann. A formalization of the security features of physical functions. In *IEEE Symposium on Security and Privacy*, pages 397–412. IEEE Computer Society, 2011. 26

[4] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Berk Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 685–702. Springer, 2009. 26

[5] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001. 168, 169

[6] Boaz Barak, Ron Canetti, Jesper B. Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *Foundations of Computer Science (FOCS'04)*, pages 394–403, 2004. 25

[7] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS*, pages 116–125, 2001. 6, 163, 164, 166, 167, 168, 169

[8] Mihir Bellare, Rafael Dowsley, Brent Waters, and Scott Yilek. Standard security does not imply security against selective-opening. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 645–662. Springer, 2012. 91

[9] Mihir Bellare, Marc Fischlin, Shafi Goldwasser, and Silvio Micali. Identification protocols secure against reset attacks. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 495–511. Springer, 2001. 164, 165, 183, 184

[10] Mihir Bellare and Oded Goldreich. On Defining Proofs of Knowledge. In *Advances in Cryptology – Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer Verlag, 1993. 15, 167

[11] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT*, pages 1–35, 2009. 10, 11, 90, 91, 93, 95, 96

[12] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1986. 12, 16, 17, 133, 142, 168, 174

[13] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988. 58

[14] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 51–70. Springer, 2011. 4, 21, 26, 28, 29, 31, 33, 34, 37

[15] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. *IACR Cryptology ePrint Archive*, 2011:681, 2011. 30

[16] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science (FOCS'01)*, pages 136–145, 2001. 2, 9, 17, 19, 20, 28, 36

[17] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007. 21, 26, 28, 34, 85

[18] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001. 21, 25

[19] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000. 5, 6, 121, 122, 144, 163, 164, 165, 168

[20] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-Box Concurrent Zero-Knowledge Requires $\omega(\log n)$ Rounds. In *33st ACM Symposium on Theory of Computing (STOC '01)*, pages 570–579. ACM, 2001. 111

[21] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer, 2003. 3, 21, 25

[22] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In John H. Reif, editor, *STOC*, pages 494–503. ACM, 2002. 25, 28, 37, 41

[23] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for uc secure computation using tamper-proof hardware. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 545–562. Springer, 2008. 25, 56, 58

[24] Chongwon Cho, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Simultaneously resettable arguments of knowledge. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 530–547. Springer, 2012. 6, 124

[25] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (efficient) universally composable two-party computation using a minimal number of stateless tokens. *IACR Cryptology ePrint Archive*, 2011:689, 2011. 25, 58

[26] Ronald Cramer, Ivan Damgrard, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology – Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994. 15, 16, 137, 155, 156

[27] Giovanni Di Crescenzo and Rafail Ostrovsky. On concurrent zero-knowledge with pre-processing. In *CRYPTO*, pages 485–502, 1999. 38, 93

[28] Ivan Damgaard and Alessandra Scafuro. Unconditionally secure and universally composable commitments from physical assumptions. In submission to Eurocrypt 2013. 4

[29] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, pages 251–260. IEEE Computer Society, 2009. 6, 163, 164, 165, 168, 169, 170

[30] Giovanni Di Crescenzo. Minimal assumptions and round complexity for concurrent zero-knowledge in the bare public-key model. In *COCOON*, pages 127–137, 2009. 122, 128, 132, 137, 138

[31] Giovanni Di Crescenzo, Giuseppe Persiano, and Ivan Visconti. Constant-round resettable zero knowledge with concurrent soundness in the bare public-key model. In *Advances in Cryptology – Crypto '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 237–253. Springer-Verlag, 2004. 122, 132, 133, 163

[32] Giovanni Di Crescenzo, Giuseppe Persiano, and Ivan Visconti. Improved Setup Assumptions for 3-Round Resettable Zero Knowledge. In *Asiacrypt '04*, volume 3329 of *Lecture Notes in Computer Science*, pages 530–544. Springer-Verlag, 2004. 124

[33] Giovanni Di Crescenzo and Ivan Visconti. Concurrent zero knowledge in the public-key model. In *Proc. of ICALP '05*, volume 3580 of *Lecture Notes in Computer Science*, pages 22–33. Springer, 2005. 122, 132, 134, 135

[34] Giovanni Di Crescenzo and Ivan Visconti. On defining proofs of knowledge in the bare public key model. In *ICTCS*, pages 187–198, 2007. 122, 135

[35] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008. 30

[36] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Ishai [60], pages 164–181. 25

[37] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. David & goliath oblivious affine function evaluation - asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token. Cryptology ePrint Archive, Report 2012/135, 2012. `http://eprint.iacr.org/`. 25

[38] Cynthia Dwork and Moni Naor. Zaps and their applications. In *In 41st FOCS*, pages 283–293. IEEE, 2000. 168

[39] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry Stockmeyer. Magic functions. In *Foundations of Computer Science (FOCS'99)*, pages 523–534, 1999. 4, 89, 90

[40] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *Proceedings of the 20th annual ACM symposium on Theory of computing*, STOC '98, pages 409–418. ACM, 1998. 93, 121

[41] Ilze Eichhorn, Patrick Koeberl, and Vincent van der Leest. Logically reconfigurable pufs: memory-based secure key storage. In *Proceedings of the sixth ACM workshop on Scalable trusted computing*, STC '11, pages 59–64, New York, NY, USA, 2011. ACM. 26

[42] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999. 133, 170

[43] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 526–544. Springer, 1989. 12, 137

[44] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *in 22nd STOC*, pages 416–426. ACM Press, 1990. 15

[45] Sanjam Garg, Rafail Ostrovsky, Ivan Visconti, and Akshay Wadia. Resettable statistical zero knowledge. In *TCC*, Lecture Notes in Computer Science. Springer-Verlag, 2012. 163

[46] Rosario Gennaro and Silvio Micali. Independent zero-knowledge sets. In *ICALP*, volume 4052 of *Lecture Notes in Computer Science*, pages 181–234. Springer, 2006. 89

[47] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001. 19

[48] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for np. *J. Cryptology*, 9(3):167–190, 1996. 92, 176, 177

[49] Shafi Goldwasser, Yael T. Kalai, and Guy. N. Rothblum. One-time programs. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56. Springer, Berlin, Germany, 2008. 25

[50] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984. 1, 14

[51] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *STOC*, pages 291–304. ACM, 1985. 2, 163

[52] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. 14, 19

[53] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 173–190. Springer, 2010. 56, 58, 79, 80

[54] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2010. 25, 58

[55] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 51–60. IEEE Computer Society, 2012. 94

[56] Vipul Goyal and Hemanta K. Maji. Stateless cryptographic protocols. In *FOCS*, 2011. 165

[57] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *Advances in Cryptology – CRYPTO 06*, volume 4117 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2006. 168

[58] Jorge Guajardo, Sandeep S. Kumar, Geert Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2007. 26

[59] Dennis Hofheinz. Possibility and impossibility results for selective decommitments. *J. Cryptology*, 24(3):470–516, 2011. 10, 11, 62, 90, 91, 95, 96, 97

[60] Yuval Ishai, editor. *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*. Springer, 2011. 190, 195

[61] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *37th Annual ACM Symposium on Theory of Computing*, pages 644–653, 2005. 25

[62] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007. 3, 25, 26, 56, 57, 58

[63] Stefan Katzenbeisser, Ünal Koccabas, Vladimir Rozic, Ahmad-Reza Sadeghi, Ingrid Verbauwhede, and Christian Wachsmann. Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In Prouff and Schaumont [89], pages 283–301. 26

[64] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, STOC '92, pages 723–732, New York, NY, USA, 1992. ACM. 59

[65] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in polyloalgorithm rounds. In *Proceedings of the 33rd annual ACM symposium on Theory of computing*, STOC '01, pages 560–569. ACM, 2001. 132, 163

[66] Ünal Koccabas, Ahmad-Reza Sadeghi, Christian Wachsmann, and Steffen Schulz. Poster: practical embedded remote attestation using physically unclonable functions. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 797–800. ACM, 2011. 26

[67] Vladimir Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 327–342, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany. 58

[68] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC*, pages 683–692, 2003. 112, 116

[69] Philip D. MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In *EUROCRYPT'04*, pages 382–400, 2004. 11

[70] Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In Prouff and Schaumont [89], pages 302–319. 26

[71] Roel Maes and Ingrid Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 3–37. Springer Berlin Heidelberg, 2010. 26

[72] Silvio Micali and Leonid Reyzin. Min-round resettable zero-knowledge in the public-key model. In *Advances in Cryptology – Eurocrypt '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 373–393. Springer-Verlag, 2001. 124

[73] Silvio Micali and Leonid Reyzin. Soundness in the public-key model. In *Advances in Cryptology – Crypto '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 542–565. Springer-Verlag, 2001. 122, 125, 132, 133, 163

[74] Daniele Micciancio and Erez Petrank. Simulatable commitments and efficient concurrent zero-knowledge. In *Advances in Cryptology – Eurocrypt '03*, volume 2045 of *Lecture Notes in Computer Science*, pages 140–159. Springer-Verlag, 2003. 134

[75] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 527–544, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany. 25

[76] Moni Naor. Bit commitment using pseudo-randomness. In *CRYPTO*, pages 128–136, 1989. 38, 70, 72

[77] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991. 12

[78] Rafail Ostrovsky, Giuseppe Persiano, and Ivan Visconti. Constant-round concurrent non-malleable zero knowledge in the bare public-key model. In *Proc. of ICALP '08*, volume 5126 of *Lecture Notes in Computer Science*, pages 548–559. Springer, 2008. 124

[79] Rafail Ostrovsky, Vanishree Rao, Alessandra Scafuro, and Ivan Visconti. Revisiting lower and upper bounds for selective decommitments. In submission to TCC 2013. 5, 123

[80] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In submission to Eurocrypt 2013. 4, 55

[81] Ravikanth S. Pappu, Ben Recht, Jason Taylor, and Niel Gershenfeld. Physical one-way functions. *Science*, 297:2026–2030, 2002. 25

[82] Ravikanth Srinivasa Pappu. *Physical One-Way Functions*. PhD thesis, MIT, 2001. 25

[83] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *Proceedings of the 37th annual ACM symposium on Theory of computing*, STOC '05, pages 533–542. ACM, 2005. 168, 169, 171, 174

[84] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In *TCC*, pages 403–418, 2009. 13, 14, 91, 106

[85] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, 1992. Springer-Verlag. 11, 12, 97, 101, 155

[86] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Cynthia Dwork, editor, *STOC*, pages 187–196. ACM, 2008. 36

[87] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *In 43rd FOCS*, pages 366–375, 2002. 111, 123, 132

[88] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *36th Annual ACM Symposium on Theory of Computing*, pages 242–251, 2004. 25

[89] Emmanuel Prouff and Patrick Schaumont, editors. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*. Springer, 2012. 192, 193

[90] Leonid Reyzin. *Zero-Knowledge with Public Keys, Ph.D. Thesis*. MIT, 2001. 11, 12, 133

[91] Ransom Richardson and Joe Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 415–431. Springer-Verlag, 1999. 123, 132

[92] Ulrich Rührmair. Oblivious transfer based on physical unclonable functions. In Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi, editors, *TRUST*, volume 6101 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 2010. 26

[93] Ulrich Rührmair, Stefan Katzenbeisser, and H. Busch. Strong pufs: Models, constructions and security proofs. In A. Sadeghi and P. Tuyls, editors, *Towards Hardware Intrinsic Security: Foundations and Practice*, pages 79–96. Springer, 2010. 26

[94] Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. Enhancing rfid security and privacy by physically unclonable functions. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 281–305. Springer Berlin Heidelberg, 2010. 26

[95] Alessandra Scafuro and Ivan Visconti. On round-optimal zero knowledge in the bare public-key model. In *EUROCRYPT*, Lecture Notes in Computer Science. Springer-Verlag, 2012. 5, 163

[96] Claus-Peter Schnorr. Efficient Signature Generation for Smart Cards. *Journal of Cryptology*, 4(3):239–252, 1991. 16, 137, 155

[97] Pim Tuyls and Lejla Batina. Rfid-tags for anti-counterfeiting. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 115–131. Springer, 2006. 26

[98] Margarita Vald. Personal communication. 34

[99] Ivan Visconti. Efficient zero knowledge on the internet. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 22–33. Springer, 2006. 122, 132, 134, 135

[100] David Xiao. (nearly) round-optimal black-box constructions of commitments secure against selective opening attacks. In Ishai [60], pages 541–558. 5, 90, 91, 92, 93, 95, 96, 97, 111, 112

[101] David Xiao. On the round complexity of black-box constructions of commitments secure against selective opening attacks. Cryptology ePrint Archive, Report 2009/513 - Revision May 29, 2012, 2012. http://eprint.iacr.org/. 91, 92

[102] David Xiao. Round-optimal black-box statistically binding selective-opening secure commitments. In *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*, volume 7374 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2012. 91, 92

[103] Andrew Chi-Chih Yao, Moti Yung, and Yunlei Zhao. Concurrent knowledge-extraction in the public-key model. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(002), 2007. 122, 128, 137

[104] Andrew Chi-Chih Yao, Moti Yung, and Yunlei Zhao. Concurrent knowledge extraction in the public-key model. In *ICALP (1)*, pages 702–714, 2010. 122, 128, 132, 137

[105] Deng Yi, Dengguo Feng, Vipul Goyal, Dongdai Lin, Amit Sahai, and Moti Yung. Resettable cryptography in constant rounds - the case of zero knowledge. In *ASIACRYPT*, 2011. 124, 164, 165, 170

[106] Moti Yung and Yunlei Zhao. Generic and practical resettable zero-knowledge in the bare public-key model. In *EUROCRYPT*, pages 129–147, 2007. 122, 128, 132, 136, 163

[107] Yunlei Zhao. Concurrent/resettable zero-knowledge with concurrent soundness in the bare public-key model and its applications. Cryptology ePrint Archive, Report 2003/265, 2003. `http://eprint.iacr.org/`. 122, 128, 132, 136

[108] Yunlei Zhao, Xiaotie Deng, Chan H. Lee, and Hong Zhu. Resettable zero-knowledge in the weak public-key model. In *Advances in Cryptology – Eurocrypt '03*, volume 2045 of *Lecture Notes in Computer Science*, pages 123–139. Springer-Verlag, 2003. 124