University of California

Los Angeles

Secure Computation from Hardware Assumptions

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Akshay Wadia

2014

Abstract of the Dissertation

Secure Computation from Hardware Assumptions

by

Akshay Wadia

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Rafail Ostrovsky, Co-chair

Professor Amit Sahai, Co-chair

Highly concurrent environments, like the Internet, present new challenges towards design of secure cryptographic protocols. Indeed, it is known that protocols proved secure in the so called 'stand-alone' model, where a protocol is assumed to execute in isolation, are no longer secure in a concurrent environment. In fact, the case of arbitrary composition is so severe that no security can be achieved without an external secure set-up. Numerous such set-ups have been proposed in the literature, each with its own advantages and disadvantages. In this thesis, we study two new set-ups motivated by recent advances in secure hardware design: tamper-proof tokens, and physically uncloneable functions. For both set-ups, we provide universally composable protocols for general cryptographic tasks. Additionally, our protocols using tamper-proof tokens are information-theoretically secure, and non-interactive.

The dissertation of Akshay Wadia is approved.

Rafail Ostrovsky, Committee Co-chair

Amit Sahai, Committee Co-chair

Vwani Roychowdhury

Eleazar Eskin

University of California, Los Angeles

2014

Table of Contents

List of Figures

Acknowledgments

First and foremost, I want to express my gratitude towards my advisors, Rafail Ostrovsky and Amit Sahai.

One never leaves Rafi's office without a basketful of new ideas. He has this great ability to generate many different ideas to tackle a problem, and then quickly prune them to the few that are the most promising. He is also masterful at taking tools that are seemingly unrelated to a problem, and then molding them to crack the puzzle, and I hope I have absorbed some of this skill. In addition to his technical guidance, Rafi has been extremely supportive of my ideas and decisions about various things, and I am very grateful for that.

Amit's clarity of thought and depth of understanding of research and life are truly inspiring. Many a times I have walked into his office with a mess of confused ideas, which he has deftly untangled, and then proceeded to patiently explain to me my own thoughts. Even on things that I feel I understand well, talking to him has often lead to profound change in perspective. Many times during my PhD, in times of confusion and uncertainty, Amit has been an unfailing source of helpful advice. I have always felt much more confident about my decisions after vetting them by him.

I would also like to thank Vwani and Zar for being on the committee. I am grateful to Vwani for giving me an opportunity to work in his group, and hopefully this will lead to interesting things in the future. Hanging out briefly in Zar's lab was a lot of fun, and gave me in interesting glimpse into the field of computational genetics.

I would also like to thank Krzysztof Pietrzak for hosting me at IST Austria for a summer. It was a great experience, particularly because Krzysztof is so fun to work with. I learnt a lot from him over the summer, including how to keep a good balance between research and other parts of life.

I should also thank my colleagues who made up the menagerie that was the theory and crypto lab at UCLA: Shweta, Prabhanjan, Nishanth, Chongwon, Sanjam, Shankar, Vipul, Divya, Abhishek, Bhavana, Dakshita, Abishek, James, Maji, Omkant, Anat, Vanishree, Alan, Alessandra, Michael, Ivan. The entire bunch has had a profound influence on me. Whether this influence has been deleterious or salubrious, remains to be seen.

I want to thank Prathibha and Jacob. Their company has unwaveringly provided refuge and refreshment in time of need, which was most of the time. Discussions with Jacob on various subjects, although wine infused on most occasions, have sharpened my thinking perceptibly.

Lastly, and certainly not leastly, I want to thank my family. Their contribution in all of this is so significant and primary, that it is laughable to even attempt to summarize it here. So I won't bother.

Vita

2003	Bachelor of Information Technology, University of Delhi, New Delhi, India
2006	MS, Computer Science, University at Buffalo, SUNY, New York, USA

Publications

V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, A. Wadia, "Founding Cryptography on Tamper-Proof Hardware Tokens", *TCC 2010.*

S. Garg, R. Ostrovsky, I. Visconti, A. Wadia, "Resettable Statistical Zero-Knowledge", *TCC 2012.*

A. Kumarasubramanian, R. Ostrovksy, O. Pandey, A. Wadia, "Cryptography from CAPTCHAs", *PKC 2013.*

S. Krenn, K. Pietrzak, A. Wadia, "A Counterexample to the Chain Rule of Conditional HILL Entropy - And What Deniable Encryption Has to Do With It", *TCC 2013.*

R. Ostrovsky, A. Scafuro, I. Visconti, A. Wadia, "Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions", *Eurocrypt 2013.*

R. Gelles, A. Sahai, A. Wadia, "Private Interactive Coding over an Adversarial Channel", *ITCS 2014.*

D. He, N. A. Furlotte, F. Hormozdiari, J. W. J. Joo, A. Wadia, R. Ostrovsky, A. Sahai, E. Eskin, “Identifying Genetic Relatives Without Compramising Privacy”, *Genome Research 2014.*

F. Hormozdiari, J. W. J. Joo, A. Wadia, F. Guan, R. Ostrovsky, A. Sahai, E. Eskin, “Privacy Preserving Protocol for Detecting Genetic Relatives Using Rare Variants”, *Bioinformatics 30(12), 2014.*

M. Prabhakaran, A. Sahai, A. Wadia, “Cryptography Using Leaky-Tokens”, *ICALP 2014.*

CHAPTER 1

Introduction

Consider two parties, Alice and Bob, each of whom holds private inputs x_A and x_B, respectively. They wish to compute a joint function $f(x_A, x_B) = (y_A, y_B)$ over their inputs in such a way, that the other party does not learn anything about their private inputs. This is known as the problem of *secure computation*. This problem has attracted tremendous attention over the history of cryptographic research, and continues to occupy a marquee position in the field. Various axes along which secure computation has been studied include the kind of security obtained (for eg., computational vs. information theoretic), complexity of the algorithm that the parties run, number of rounds in the protocol, etc. An important and well studied generalization of the problem is to the setting of more than two parties, but we will focus exclusively on the two party case in this thesis. A good discussion of various introductory results in this area can be found in [Gol01, Gol04].

The classical setting in which secure computation was studied is the so called *stand-alone* setting. In this setting, it is assumed that the protocol is being run in isolation. Research in this setting culminated in the discovery of general protocols for any polynomial-time functionality [GMW87]. However, the assumption of isolation is not tenable in the presence of highly concurrent environments like the Internet. For example, on your laptop, you may be logging in to your online banking account, while chatting with a friend on a messenger service, while checking your mail. All these protocols are running concurrently and with arbitrary interleaving on your laptop. Thus, there is a need to analyze protocols in such a concurrent setting.

A formal framework to study security in a concurrent setting was described by Canetti [Can01a]. This notion of security is known as Universally Composable (UC) security. It was soon observed that stand-alone security is not sufficient for UC security. In fact, it was shown by Canetti and Fischlin [CF01] that UC security is impossible to obtain without *set-up* assumptions. A set-up is a functionality that both parties have access to during protocol execution. It is assumed that this functionality is provided to the protocol externally, and its security properties do not need to be established within the protocol. For example, one of the first set-up assumptions considered was the so called Common Reference String (CRS). The CRS functionality simply samples a string of appropriate length from a pre-specified distribution and sends it to both the parties. Given this simple functionality, it can be shown how UC security can be achieved.

Given that the formal security proof simply assumes the security of the set-up, it is natural to ask which set-ups are valid to consider. For instance, what prevents us from choosing a set-up that is so strong, that it trivializes the task of protocol construction? Because of these issues, we focus only on those set-ups that have strong heuristic guarantees in real life. Consider a variant of the CRS set-up, known as the Common *Random* String model, where the underlying distribution is uniform over the set of string of appropriate length. There are many proposed heuristics to implement this set-up. For instance, taking the average of the stock price of a particular stock over the past ten years as the random string can be considered heuristically uniform. Another proposal is to measure variations in solar activity over a period of time.

The heuristic nature of set-ups strongly motivates the discovery and investigation of a variety of different set-up assumptions. Motivated by developments in secure hardware manufacturing, researchers have recently considered set-ups based on hardware assumptions. These set-ups involve parties using and physically exchanging actual hardware devices during the course of a protocol. In this thesis, we will study two hardware based set-up assumptions, and show how UC security can be achieved using them. The set-ups we consider are:

Tamper-proof Hardware Tokens: These are hardware boxes that a sender can initialize with any polynomial time functionality. The tamper-proof property guarantees that the receiver has access only to the input/output behaviour of the functionality, and can not learn anything else about the functionality.

Physically Uncloneable Functions: These are hardware boxes that provide an implementation of a random function. Further, the randomness is obtained from complex physical properties of materials used in the construction of these boxes. This prevents an adversary from cloning such a box, and thus a party can evaluate the box only when it is in its possession.

The organization of the thesis is as follows. In Chapter 2, we formally model tamper-proof tokens. In Chapter 3, we give a non-interactive unconditional UC secure protocol for any functionality. This protocol relies on *stateful* tokens, that is, tokens which retain state over execution. In Chapter 4, we consider the simpler stateless tokens, and give a UC protocol that relies on cryptographic assumptions. In Chapter 5 we introduce Physically Uncloneable Functions, and provide the formal model. Finally, in Chapter 6 we provide UC secure protocol for general functionalities in the Physically Uncloneable Function model.

CHAPTER 2

Modeling Tamper-Proof Hardware

Our model for tamper-proof hardware is similar to that of Katz ([Kat07]). However, as we consider both stateful and stateless tokens, we define different ideal functionalities for the two. Here, we formally define the ideal functionalities $\mathcal{F}_{wrap}^{single}$, for single use stateful tokens, and $\mathcal{F}_{wrap}^{stateless}$ for stateless tokens.

The functionality $\mathcal{F}_{wrap}^{single}$ is used to model hardware tokens that can be executed only once. Thus, the only state these tokens keep is a flag which indicates whether the token has been run or not. To model this behaviour, $\mathcal{F}_{wrap}^{single}$ deletes the token from its memory after it has been run once. The formal description of $\mathcal{F}_{wrap}^{single}$ is presented in Figure 2.1.

Next, we define the functionality $\mathcal{F}_{wrap}^{stateless}$ that models stateless tokens. The idea of $\mathcal{F}_{wrap}^{stateless}$, described in Figure 2.2, is to model the following real-world functionality: party P_i sends a stateless token M to party P_j. Since the token is stateless, P_j can run M multiple times on inputs of its choice. Thus, $\mathcal{F}_{wrap}^{stateless}$ saves the description of the Turing machines it gets from a party in create messages, and lets the other party run them multiple times. Each machine is uniquely identified by a machine identifier mid.

One can also consider a variant of $\mathcal{F}_{wrap}^{stateless}$ which allows a malicious sender to generate *stateful* tokens. Our protocols which use stateless tokens are secure in this more adversarial setting as well. (This is automatically the case in all protocols for which an honest receiver makes only a single use of each token.)

We are interested in non-interactive protocols in which the communication involves a single batch of tokens sent from a "sender" to a "receiver". (One could also allow the sender to send a message to a receiver; however, from a feasibility point of view this could also be done in the simpler model in which only tokens are sent.)

Functionality $\mathcal{F}_{wrap}^{single}$

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

Create Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid}, M)$ from P_i, where M is a Turing machine and mid is machine id, do:

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

Execute Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, \mathsf{mid}, \mathsf{msg})$ from P_j, find the unique stored tuple $(P_i, P_j, \mathsf{mid}, M)$ (if no such tuple exists, do nothing). Choose random $r \leftarrow \{0,1\}^{p(k)}$. Run $M(\mathsf{msg}; r)$ for at most $p(k)$ steps, and let out be the response ($\mathsf{out} = \perp$ if M does not halt in $p(k)$ steps). Send $(\mathsf{sid}, P_i, \mathsf{mid}, \mathsf{out})$ to P_j, and delete $(P_i, P_j, \mathsf{mid}, M)$.

Figure 2.1: Ideal functionality for single-use stateful tokens

Definition 1. *(Non-Interactive Protocols in the Tamper-Proof Hardware Model) A two-party protocol $\Pi = (P_1, P_2)$ is **non-interactive** if the only messages sent by P_1 are* create *messages to $\mathcal{F}_{wrap}^{single}$ (or $\mathcal{F}_{wrap}^{stateless}$) and the only messages sent by P_2 are* run *messages to $\mathcal{F}_{wrap}^{single}$ (or $\mathcal{F}_{wrap}^{stateless}$).*

2.1 One-Time Programs

Informally, a *one-time program* (OTP) [GKR08] for a function f lets a party evaluate f on only one input chosen by that party at run time. The intuitive security goal is that no efficient adversary, after evaluating the one-time program on x, can learn anything about $f(y)$ for some $y \neq x$, other than what can be inferred from $f(x)$. In our constructions, OTPs will be used within other protocols, thus it would be convenient for us to view them as two-party non-interactive protocols in the hardware token model, which are secure against malicious receivers. We thus view OTP as implementing a two-party functionality $f(\cdot,\cdot)$

Functionality $\mathcal{F}_{wrap}^{stateless}$

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

Create Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid}, M)$ from P_i, where M is a Turing machine, do:

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

Execute Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, \mathsf{mid}, \mathsf{msg})$ from P_j, find the unique stored tuple $(P_i, P_j, \mathsf{mid}, M)$. If no such tuple exists, do nothing. Run $M(\mathsf{msg})$ for at most $p(k)$ steps, and let out be the response ($\mathsf{out} = \perp$ if M does not halt in $p(k)$ steps). Send $(\mathsf{sid}, P_i, \mathsf{mid}, \mathsf{out})$ to P_j.

Figure 2.2: Ideal functionality for stateless tokens

(where the description of f is known to both parties), where the first (secret) input is fixed by the sender during construction.

Figure 2.3 defines the ideal functionality for a one-time program for function $f(\cdot, \cdot)$.

Functionality $\mathcal{F}_f^{OTP}$

Create Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, x)$ from P_i, where x is a string, do:

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

Execute Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, y)$ from P_j, find the stored tuple (P_i, P_j, x) (if no such tuple exists, do nothing). Send $f(x, y)$ to P_j and delete tuple (P_i, P_j, x).

Figure 2.3: Ideal functionality for One-time Program for function $f(\cdot, \cdot)$.

Now we define OTPs in the $\mathcal{F}_{wrap}^{single}$-hybrid model.

Definition 2. *(One-Time Program for $f(\cdot,\cdot)$) A one-time program for function $f(\cdot,\cdot)$ is a two-party non-interactive protocol $\Pi = (P_1, P_2)$ in the $\mathcal{F}_{wrap}^{single}$-hybrid model, such that for every probabilistic polynomial time adversary $\mathcal{A}$ corrupting P_2, there exists a probabilistic polynomial time ideal-world adversary S called the simulator, such that for every environment $\mathcal{Z}$,*

$$\mathsf{IDEAL}_{S,\mathcal{Z}}^{\mathcal{F}_f^{OTP}} \sim \mathsf{REAL}_{\Pi,\mathcal{A},Z}.$$

One way of implementing OTPs using hardware tokens is to construct stateful hardware tokens that contain the entire code of the function $f(x,\cdot)$. However, like in [GKR08] we would like to use only the simplest kind of hardware tokens in our protocols. To this end, we focus on using one-time-memory (OTM) tokens only. OTM tokens realize the ideal functionality defined in Figure 2.5.

2.2 Flavors of OT

The ideal OT functionality required by unconditionally secure protocols in the OT-hybrid model [Kil88, IPS08] is denoted by $\mathcal{F}^{\mathrm{OT}}$ and is formally defined in Figure 2.4.

Functionality $\mathcal{F}^{\mathrm{OT}}$

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$.
On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c)$ to P_j, **send an acknowledgment** $(P_j, \mathsf{sid}, \mathsf{id})$ **to** P_i, and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$. Else, do nothing.

Figure 2.4: Ideal functionality for OT

Following [GKR08], we refer to simple hardware tokens that implement a single OT call as *OTM (one-time-memory)* tokens. We give the formal definition of OTM tokens here, and discuss these tokens in detail in Section 3.1. OTM tokens are defined in Figure 2.5.

Functionality OTM

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$.

On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c)$ to P_j and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$. Else, do nothing.

Figure 2.5: Ideal functionality for OTM tokens

We also define a parallel version of the OTM functionality, denoted pOTM, which allows the receiver to make several parallel OTM choice. Note that unlike a direct implementation using separate OTM tokens, this functionality requires the receiver to make all its choices at once, and does not allow a malicious receiver to determine its choice bits in an adaptive fashion. The ideal functionality for pOTM is defined in Figure 2.6.

Functionality pOTM

pOTM is parameterized by an integer k.

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k))))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k))))$.

On receiving $(P_i, \mathsf{sid}, \mathsf{id}, (c^1, \ldots, c^k))$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k)))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, (s_{c^1}^1, \ldots, s_{c^k}^k))$ to P_j and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k)))$. Else, do nothing.

Figure 2.6: Ideal functionality for parallel-OTM.

Finally, the ExtOTM functionality differs from OTM in that it takes an additional input r from the sender, and delivers this input to the receiver together with its chosen string s_c. This functionality is described in Figure 2.7. It is easy to see that a protocol for the ExtOTM functionalities allows us to realize the $\mathcal{F}^{\mathrm{OT}}$functionality as required by [Kil88, IPS08].

Functionality ExtOTM

On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0, s_1), r))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0, s_1), r))$.
On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, ((s_0, s_1), r))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c, r)$ to P_j and delete the tuple $(P_i, P_j, id, ((s_0, s_1), r))$. Else, do nothing.

Figure 2.7: Ideal functionality for ExtOTM

2.3 Other Definitions

Definition 3. *(Computational Indistinguishability) Two distribution ensembles $X := \{ X_n \}_{n \in \mathbf{N}}$ and $Y := \{ Y_n \}_{n \in \mathbf{N}}$ are* ***computationally indistinguishable*** *(written as $X \sim Y$) if for every probabilistic polynomial-time algorithm D, every positive polynomial $p(\cdot)$, and all sufficiently large n's,*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| \leq \frac{1}{p(n)}.$$

The *statistical difference* between two distribution ensembles $X := \{ X_n \}_{n \in \mathbf{N}}$ and $Y := \{ Y_n \}_{n \in \mathbf{N}}$ is defined by

$$\Delta(n) = \frac{1}{2} \sum_{\alpha} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|.$$

Ensembles X and Y are *statistically close* if the their statistical difference is negligible in n.

Now we define the next-message function of an interactive TM P:

Definition 4. *(Next Message Function) Let P be an interactive TM. The* ***next message function for round*** *i of P is the function $P_i^{x,r}()$, which on input $(m_1, \ldots, m_i, s_{i-1})$, outputs $(\hat{m}_i, s_i)$, where $\hat{m}_i$ is the message output by P on input x and random input r, after receiving $m_1, \ldots, m_i$ in previous rounds, and s_{i-1} and s_i contain auxiliary state information for rounds $i-1$ and i respectively.*

Commitments. We will use the two-round statistically binding commitment scheme com ([Nao91]). To recall, to commit to bit b, the first message in Naor's scheme is a random string r from receiver to sender. Then the sender responds with either $G(s)$ or $G(s) \oplus r$ depending of whether $b = 0$ or $b = 1$, where $G(\cdot)$ is a pseudo-random generator, and s is a randomly chosen seed. To decommit, the sender sends (b, s) to the receiver. We observe that Naor's scheme has the property that the same r can be used for committing to (polynomially) many bits. Thus, once the receiver's string has been initially exchanged, we essentially have a non-interactive commitment function. For receiver's message r, denote sender's response by $\mathsf{com}_r(b)$, where b is the bit being committed. Abusing terminology, we will call $\mathsf{com}_r(b)$ the commitment to bit b. When the randomness used in the first message is clear from the context, we will drop the subscript r, and will denote the committed string by $\mathsf{com}(b)$. The opening of a commitment α is denoted by $\mathsf{open}(\alpha)$ and consists of the committed bit b and seed s (the randomness r is implicit).

Unconditional One-Time MAC. By (MAC, VF), we will denote an unconditional one-time message authentication scheme, where $MAC_k(m)$ represents tagging message m with key k, and $VF_k(m, \sigma)$ denotes the verification algorithm, which returns 1 if σ is the correct tag of m under key k. An example of such a MAC is as follows: the key k is a pair of κ length strings (a, b) (where κ is the security parameter). The tag of message m with key $k = (a, b)$ is $a \cdot m + b$, where all operations are in $GF(2^\kappa)$.

2.3.1 The UC Model

We use the UC-framework of Canetti [Can01b] to capture the general notion of secure computation of (possibly reactive) functionalities. Our main focus is on the two-party case. We will usually refer to one party as a "sender" and to another as a "receiver". A *non-reactive* functionality may receive an input from each party and deliver output to each party (or only to the receiver). A *reactive* functionality may have several rounds of inputs and outputs, possibly maintaining state information between rounds. We begin by defining protocol syntax, and then informally review the UC-framework. For more details, see [Can01b].

Protocol syntax. Following [GMR89] and [Gol01], 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. This is called the *real-life* model. Finally, an *ideal process* is considered, where the parties only interact with the ideal functionality, and not amongst themselves. Informally, a protocol realizes an ideal functionality if running of the protocol amounts to "emulating" the ideal process for that functionality.

Let $\Pi = (P_1, P_2)$ be a protocol, and $\mathcal{F}$ be the ideal-functionality. We describe the ideal and real world executions.

The real-life model. The real-life model consists of the two parties P_1 and P_2, the environment $\mathcal{Z}$, and the adversary $\mathcal{A}$. Adversary $\mathcal{A}$ can communicate with environment $\mathcal{Z}$ and can corrupt any party. When $\mathcal{A}$ corrupts party P_i, it learns P_i's entire internal state, and takes complete control of P_i's input/output behaviour. The environment $\mathcal{Z}$ sets the parties' initial inputs. Let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ be the distribution ensemble that describes the environment's output when protocol Π is run with adversary $\mathcal{A}$.

The ideal process. The ideal process consists of two "dummy parties" $\hat{P}_1$ and $\hat{P}_2$, the ideal functionality $\mathcal{F}$, the environment $\mathcal{Z}$, and the ideal world adversary S, called the simulator. In the ideal world, the uncorrupted dummy parties obtain their inputs from environment $\mathcal{Z}$ and simply hand them over to $\mathcal{F}$. As in the real world, adversary S can corrupt any party. Once it corrupts party $\hat{P}_i$, it learns $\hat{P}_i$'s input, and takes complete control of its input/output behaviour. Let $\mathsf{IDEAL}^{\mathcal{F}}_{S,\mathcal{Z}}$ be the distribution ensemble that describes the environment's output in the ideal process.

Definition 5. *(Realizing an Ideal Functionality) Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality, and Π be a protocol. We say Π* ***realizes*** *$\mathcal{F}$ if for any real-world adversary $\mathcal{A}$, there exists an ideal process adversary S such that for every environment $\mathcal{Z}$,*

$$\mathsf{IDEAL}^{\mathcal{F}}_{S,\mathcal{Z}} \sim \mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}.$$

CHAPTER 3

Unconditional Non-Interactive Secure Computation Using Stateful Tokens

In this chapter we establish the feasibility of unconditionally non-interactive secure computation based on *stateful* hardware tokens. As is typically the case for unconditionally secure protocols, our protocols are in fact *UC secure*.

This section is organized as follows. In 3.1 we present as a "warmup" an interactive protocol for arbitrary functionalities, which requires the parties to engage in multiple rounds of interaction. This section will introduce some useful building blocks that are used in the next . This gives an unconditional version of previous protocols for UC-secure computation based on hardware tokens [Kat07, CGS08, MS08], which all relied on computational assumptions.[1]

In 3.2 we consider the case of secure evaluation of two-party functionalities which deliver output to only one of the parties (the "receiver"). We strengthen the previous result in two ways. First, we show that in this case interaction can be completely eliminated: it suffices for the sender to non-interactively send tokens to the receiver, without any additional communication. Second, we show that even very simple, constant-size stateful tokens are sufficient for this purpose. This strengthens previous feasibility results for one-time programs [GKR08] by providing *unconditional* security (in fact, UC-security) and by offering general protection against *malicious senders*.

[1]The work of [MS08] realizes an unconditionally UC-secure *commitment* from stateful tokens. This does not directly yield protocols for secure computation without additional computational assumptions.

3.1 Warmup: The Interactive Setting

Unconditionally secure two-party computation is impossible to realize for most nontrivial functionalities, even with semi-honest parties [BOGW88, Kus92]. However, if the parties are given oracle access to a simple ideal functionality such as Oblivious Transfer (OT) [Rab81, EGL85], then it becomes possible not only to obtain unconditionally secure computation with semi-honest parties [GV87, GHY87, Gol04], but also unconditional *UC-security* against *malicious* parties [Kil88, IPS08]. This serves as a natural starting point for our construction.

In the OT-hybrid model, the two parties are given access to the following ideal OT functionality: the input of P_1 (the "sender") consists of a pair of k-bit strings (s_0, s_1), the input of P_2 (the "receiver") is a choice bit c, and the receiver's output is the chosen string s_c. The natural way to implement a single OT call using stateful hardware tokens is by having the sender send to the receiver a token which, on input c, outputs s_c and erases s_{1-c} from its internal state. The use of such hardware tokens was first suggested in the context of one-time programs [GKR08]. Following the terminology of [GKR08], we refer to such tokens as OTM (one-time-memory) tokens. OTM tokens were formally defined in Section 2.2.

An appealing feature of OTM tokens is their simplicity, which can also lead to better resistance against side-channel attacks (see [GKR08] for discussion). This simplicity feature served as the main motivation for using OTM tokens as a basis for one-time programs. Another appealing feature, which is particularly important in our context, is that the OTM functionality does not leave room for bad sender strategies: whatever badly formed token a malicious sender may send is equivalent from the point of view of an honest receiver to having the sender send a well-formed OTM token picked from some probability distribution. (This is not the case for tokens implementing more complex functionalities, such as 2-out-of-3 OT or the extended OTM functionality discussed below, for which badly formed tokens may not correspond to any distribution over well-formed tokens.)

Given the above, it is tempting to hope that our goal can be achieved by simply taking any unconditionally secure protocol in the OT-hybrid model, and using OTM tokens to implement OT calls. However, as observed in [GKR08], there is a subtle but important distinction between the OT-hybrid model and the OTM-hybrid model: while in the former model the sender knows the point in the protocol in which the receiver has already made its choice and received its output, in the latter model invoking the token is entirely at the discretion of the receiver. This may give rise to attacks in which the receiver adaptively invokes the OTM tokens “out of order,” and such attacks may have a devastating effect on the security of protocols even in the case of unconditional security.[2]

Attacks on simple solution ideas. A natural way to handle the above type of attacks is by having the sender pick a secret key r for each OTM and append this key to both strings (s_0, s_1) it stores in the OTM. Then, to emulate a single call to an OT oracle, the sender sends such an OTM token to the receiver, and expects the receiver to send back r before proceeding with the protocol. This approach can indeed be used to protect a semi-honest sender from out-of-order token invocations by a malicious receiver. However, it completely exposes the receiver to an attack by a malicious sender who puts a pair distinct strings (r_1, r_2) in the same OTM. Note that in the context of one-time programs where the sender is *semi-honest*, a variant of this approach [GKR08] does suffice to solve the problem. However, in the context of *malicious* senders that we consider here, the approach of [GKR08] does not suffice.

A more subtle attack, known as *selective abort*, arises if one tries to fix the problem above in naïve ways such as asking the sender to send a randomized hash of r, and then instructing the receiver to abort if the sender's message is different from the hash of the r that it received from the OTM. This would allow a malicious sender to cause the receiver to abort based on its private choice bit, which is not allowed by the ideal OT functionality (and can lead to real attacks on secrecy).

[2] To illustrate the effect of such attacks, consider the following functionality f. The functionality takes from the sender a pair of k-bit strings (x_1, x_2) and from the receiver a k-bit string y. If $y = x_1$, the functionality delivers (x_1, x_2) to the receiver, otherwise it delivers only x_1. Now, let Π be any OT-based protocol for f which consists of only one round of OTs from the sender to the receiver, where the receiver's OT choices are its input bits. (A simple protocol of this type with perfect security against a malicious receiver and a semi-honest sender is given in [Kil88] for any f in NC^1.) Modify Π into a new protocol Π' in which the sender, following the round of OT calls, reveals x_1 to the receiver. The protocol Π' is still perfectly secure against a malicious receiver in the OT-hybrid model, but if OT calls are realized by sending OTM tokens, the new protocol allows the receiver to always learn x_2 by first observing x_1 and then invoking the OTM tokens with $y = x_1$.

Extending the OTM functionality. To fix the above idea, we will realize an extended OTM functionality which takes from the sender a pair of strings (s_0, s_1) along with an auxiliary string r, takes from the receiver a choice bit c, and delivers to the receiver both s_c and r. We denote this functionality by ExtOTM (see Figure 2.7). What makes the ExtOTM functionality nontrivial to realize using hardware tokens is the need to protect the receiver from a malicious sender who may try to make the received r depend on the choice bit c while *at the same time* protecting the sender from a malicious receiver who may try to postpone its choice c until after it learns r.

Using the ExtOTM functionality, it is easy to realize a UC-style version of the OT functionality which not only delivers the chosen string to the receiver (as in the OTM functionality) but also delivers an acknowledgment to the sender. This flavour of the OT functionality, which we denote by $\mathcal{F}^{\mathrm{OT}}$(see Figure 2.4), can be realized by having the sender invoke ExtOTM with (s_0, s_1) and a randomly chosen r, and having the receiver send r to the sender. In contrast to OTM, the $\mathcal{F}^{\mathrm{OT}}$functionality allows the sender to force any subset of the OT calls to be completed before proceeding with the protocol. This suffices for instantiating the OT calls in the unconditionally secure protocols from [Kil88, IPS08].

Realizing ExtOTM using general[3] stateful tokens. As discussed above, we cannot directly use a stateful token for realizing the ExtOTM functionality, because this allows the sender to correlate the delivered r with the choice bit c. On the other hand, we cannot allow the sender to directly reveal r to the receiver, because this will allow the receiver to postpone its choice until after it learns r. In this , we present our protocol for realizing ExtOTM using stateful tokens. This protocol is non-interactive (i.e., it only involves tokens sent from the sender to the receiver) and will also be used as a building block towards the stronger results in the next . We start with a detailed discussion of the intuition of the protocol and its security proof.

As mentioned above, at a high level, the challenge we face is to prevent unwanted correlations in an information-theoretic way for both malicious senders and malicious receivers.

[3] **Here, we make use of general tokens. Later in this section, we will show how to achieve the ExtOTM functionality (and in fact every poly-time functionality) using only very simple tokens – just bit OTM tokens.**

This is a more complex situation than a typical similar situation where only one side needs to be protected against (c.f. [Kil90, LP07]). To accomplish this goal, we make use of secret-sharing techniques combined with additional token-based "verification" techniques to enforce honest behavior.

Our ExtOTM protocol Π_{ExtOTM} starts by having the sender break its auxiliary string r into $2k$ additive shares r^i, and pick $2k$ pairs of random strings (q_0^i, q_1^i). (Each of the strings q_b^i and r^i is k-bit long, where k is a statistical security parameter.) It then generates $2k$ OTM tokens, where the i-th token contains the pair $(q_0^i \circ r^i, q_1^i \circ r^i)$. Note that a malicious sender may generate badly formed OTM tokens which correlate r^i with the i-th choice of the receiver; we will later implement a token-based verification strategy that convinces an honest receiver that the sender did not cheat (too much) in this step.

Now the receiver breaks its choice bit c into $2k$ additive shares c^i, and invokes the $2k$ OTM tokens with these choice bits. Let $(\hat{q}^i, \hat{r}^i)$ be the pair of k-bit strings obtained by the receiver from the i-th token. Note that if the sender is honest, the receiver can already learn r. We would like to allow the receiver to learn its chosen string s_c while convincing it that the sender did not correlate all of the auxiliary strings $\hat{r}^i$ with the corresponding choice bits c_i. (The latter guarantee is required to assure an honest receiver that $\hat{r} = \sum \hat{r}^i$ is independent of c as required.)

This is done as follows. The sender prepares an additional single-use hardware token which takes from the receiver its $2k$ received strings $\hat{q}^i$, checks that for each $\hat{q}^i$ there is a valid selection $\hat{c}_i$ such that $\hat{q}^i = q^i_{\hat{c}_i}$ (otherwise the token returns $\perp$), and finally outputs the chosen string $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$. (All tokens in the protocol can be sent to the receiver at one shot.) Note that the additive sharing of r in the first $2k$ tokens protects an honest sender from a malicious receiver who tries to learn $s_{\hat{c}}$ where $\hat{c}$ is significantly correlated with r, as it guarantees that the receiver effectively commits to c before obtaining any information about r. The receiver is protected against a malicious sender because even a badly formed token corresponds to some (possibly randomized) ideal-model strategy of choosing (s_0, s_1).

Finally, we need to provide to the receiver the above-mentioned guarantee that a malicious sender cannot correlate the receiver's auxiliary output $\hat{r} = \sum \hat{r}^i$ with the choice bit c. To explain this part, it is convenient to assume that both the sender and the badly formed tokens are deterministic. (The general case is handled by a standard averaging argument.) In such a case, we call each of the first $2k$ tokens well-formed if the honest receiver obtains the same r^i regardless of its choice c^i, and we call it badly formed otherwise. By the additive sharing of c, the only way for a malicious sender to correlate the receiver's auxiliary output with c is to make *all* of the first $2k$ tokens badly formed. To prevent this from happening, we require the sender to send a final token which proves that it knows all of the $2k$ auxiliary strings $\hat{r}^i$ obtained by the receiver. This suffices to convince the receiver that not all of the first $2k$ tokens are badly formed. Note, however, that we cannot ask the sender to send these $2k$ strings r^i in the clear, since this would (again) allow a malicious receiver to postpone its choice c until after it learns r.

Instead, the sender generates and sends a token which first verifies that the receiver knows r (by comparing the receiver's input to the k-bit string r) and only then outputs all $2k$ shares r^i. The verification step prevents correlation attacks by a malicious receiver. The final issue to worry about is that the string r received by the token (which may be correlated with the receiver's choices c_i) does not reveal to the sender enough information to pass the test even if all of its first $2k$ tokens are badly formed. This follows by a simple information-theoretic argument: in order to pass the test, the token must correctly guess *all* $2k$ bits c_i, but this cannot be done (except with $2^{-\Omega(k)}$ probability) even when given arbitrary k bits of information about the c_i. We describe the protocol formally now.

Protocol Π_{ExtOTM}**.**

- ***Input:*** *P_1 gets as input k-bit strings (s_0, s_1, r), and P_2 gets as input a bit c.*

- ***Specified Output:*** *P_2 should receive (s_c, r).*

  1. *P_1 chooses $4k$ distinct strings of length k, $((s_0^1, s_1^1), \ldots, (s_0^{2k}, s_1^{2k}))$. Now P_1 chooses another $2k-1$ strings of length k, $(\rho^1, \ldots, \rho^{2k-1})$, and sets ρ^{2k} such that $\bigoplus_{i=1}^{2k} \rho^i = r$. Then,*

(a) *For* $1 \leq i \leq 2k$*, send* $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}, M^i)$ *to* $\mathcal{F}_{wrap}^{single}$*, where* M^i *implements the following functionality: on input bit* c^i*, output* $(s^i_{c^i}, \rho^i)$*.*

(b) P_1 *constructs and sends an unconditional OTP for the following functionality* F_1*:*

. *Receive input* $(\hat{s}^1, \ldots, \hat{s}^{2k})$*. For* $1 \leq i \leq 2k$*, let* $\hat{c}^i \in \{0, 1\}$ *be such that* $\hat{s}^i = s^i_{\hat{c}^i}$*. If no such* $\hat{c}^i$*s exist, output* $\perp$*. Else output* $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$*.*

(c) P_1 *constructs and sends an unconditional OTP for the following functionality* F_2*:*

. *On input* $\rho \in \{0, 1\}^k$*, check if* $\rho = \bigoplus_{i=1}^{2k} \rho^i$*. If not, output* $\perp$*. Else, output* $\rho^1 \circ \ldots \circ \rho^{2k}$*.*

2. P_2 *picks random bits* $c^1, \ldots, c^{2k-1}$*, and sets* c^{2k} *such that* $c = \bigoplus_{i=1}^{2k} c^i$*. For* $1 \leq i \leq 2k$*,* P_2 *runs* M^i *with input* c^i *and obtains* $(\hat{s}^i, \rho^i)$*. It runs the OTP for* F_1 *on input* $(\hat{s}^1, \ldots, \hat{s}^{2k})$ *and obtains string* s*. If the OTP aborts, then* P_2 *sets* $s = 0^k$*. Next,* P_2 *runs OTP for* F_2 *on input* $\bigoplus_{i=1}^{2k} \rho^i$*, and obtains string* ρ*. If* $\rho \neq \rho^1 \circ \ldots \circ \rho^{2k}$*,* P_2 *aborts. Else, it outputs* $(s, \bigoplus_{i=1}^{2k} \rho^i)$*.*

Claim 6. *Protocol* Π_{ExtOTM} *realizes* ExtOTM *with statistical UC-security in the* OTM*-hybrid model.*

Proof First consider the case of malicious sender. Let $\mathcal{A}$ be an adversary controlling P_1, and let $\mathcal{Z}$ be any environment. We define the simulator S_1^{ExtOTM} as follows:

Simulator S_1^{ExtOTM}

1. Receive input (s_0, s_1, r) from $\mathcal{Z}$ for $\mathcal{A}$. Start internal simulation of $\mathcal{A}$ with the given input.
2. Receive TMs $M^1, \ldots, M^{2k}$, and OTPs for functionalities F_1 and F_2 from $\mathcal{A}$.
3. For $1 \leq i \leq 2k$, run M^i with input 0 to obtain (s_0^i, ρ_0^i). Now rewind M^i and run it with input 1 to obtain (s_1^i, ρ_1^i). If for every $1 \leq i \leq 2k$, $\rho_0^i \neq \rho_1^i$, abort.
4. Choose $2k-1$ random bits $c^1, \ldots, c^{2k-1}$, and set c^{2k} such that $\bigoplus_{i=1}^{2k} c^i = 1$. Then,
   (a) If for any index i, $s_{c^i}^i = \perp$, abort. Run the OTP corresponding to F_1 with input $(s_{c^1}^1, \ldots, s_{c^{2k}}^{2k})$ to obtain s_1. Let j be the index such that $\rho_0^j = \rho_1^j$. Run F_1 again with input $(s_{c^1}^1, \ldots, s_{c^{j-1}}^{j-1}, s_{c^j \oplus 1}^j, s_{c^{j+1}}^{j+1}, \ldots, s_{c^{2k}}^{2k})$ to obtain s_0. If the OTP aborts in either case, set that string to the default value, 0^k.
   (b) Set $r = \bigoplus_{i=1}^{2k} \rho_{c^i}^i$. Run OTP for F_2 with input r and obtain string ρ. If $\rho \neq \rho_{c^1}^1 \circ \ldots \circ \rho_{c^2k}^{2k}$, abort. Else, send (s_0, s_1, r) to ExtOTM.

We proceed to show that $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\text{ExtOTM}}_{S_1^{ExtOTM},\mathcal{Z}}$ are statistically close. Consider the following hybrids:

Hybrid $\mathcal{H}_0$: In this experiment, $\mathcal{Z}$ interacts with simulator S_1^{extOT} only. The simulator receives inputs (s_0, s_1, r) for $\mathcal{A}$ on one hand, and bit c for P_2 on the other. It then internally simulates a real execution of the protocol between $\mathcal{A}$ and P_2, and outputs whatever the simulated P_2 outputs. Clearly, $\mathcal{H}_0$ is identical to $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: In this experiment, S_1^{extOT} runs M^1 with input 0, and then rewinds M^1 and runs it with input 1 to obtain both the outputs $s_0^1 \circ \rho_0^1$ and $s_1^1 \circ \rho_1^1$. Let c^1 be P_2's query to M^1. Then, instead of running M^1, the simulator responds with $(s_{c^1}^1, \rho_{c^1}^1)$ (if M^1 outputs $\perp$ on input bit c^1, then S_1^{extOT} returns $\perp$ to P_2).

Note that in both $\mathcal{H}_0$ and $\mathcal{H}_1$, P_2 receives the same value when it queries M^1. Thus, $\mathcal{H}_0$ and $\mathcal{H}_1$ are identical.

Hybrids $\mathcal{H}_2 \ldots \mathcal{H}_{2k}$: In hybrid $\mathcal{H}_i$ for $2 \leq i \leq 2k$, the simulator extracts both the values of M^i. When P_2 queries M^i, the simulator responds without running M^i again, as it did in the case of M^1 in $\mathcal{H}_1$. As above, all these hybrids are identical.

Hybrids $\mathcal{H}_{2k+1}$: If for all i, $1 \leq i \leq 2k$, $\rho_0^i \neq \rho_1^i$, simulator aborts.

If for each i the ρ^is are different, then F_2 must guess a $2k$ length binary string. However, its input is only of length k, and thus with probability negligibly close 1, F_2 will output the wrong sequence of ρ^is, causing P_2 to abort in $\mathcal{H}_{2k}$. Thus, $\mathcal{H}_{2k}$ and $\mathcal{H}_{2k+1}$ are statistically close.

Hybrid $\mathcal{H}_{2k+2}$: S_1^{extOT} chooses $2k-1$ random bits $c^1, \ldots, c^{2k-1}$, and sets c^{2k} such that $c^{2k} = \bigoplus_{i=1}^{2k-1} c^i$. If for any index i, $s_{c^i}^i = \perp$, the simulator aborts. Else, it runs F_1 with input $(s_{c^1}^1, \ldots, s_{c^{2k}}^{2k})$, and obtains s_0. Let j be the index such that $\rho_0^j = \rho_1^j$. Now S_1^{extOT} runs F_1 with input $(s_{c^1}^1, \ldots, s_{c^{j-1}}^{j-1}, s_{c^j \oplus 1}^j, s_{c^{j+1}}^{j+1}, \ldots, s_{c^{2k}}^{2k})$ and obtains s_1. If F_1 aborts in either execution, it sets the corresponding string to the default value 0^k. Then it runs F_2 with input $r := \bigoplus_{i=1}^{2k} \rho_{c^i}^i$, and obtains output ρ. If $\rho \neq \rho_{c^1}^1 \circ \ldots \circ \rho_{c^{2k}}^{2k}$, simulator aborts. Finally, S_1^{extOT} ignores P_2 and outputs (s_c, r) as P_2's output.

Note that P_2's input bits to the M^is are $2k-1$-wise independent. Thus, as there is at least one M^i which does not abort on either input, the probability that P_2 aborts in $\mathcal{H}_{2k+1}$ (before querying F_1) is the same as the probability that S_1^{extOT} aborts in $\mathcal{H}_{2k+1}$ (before running F_1).

Next, consider the joint distribution of inputs to F_1 and F_2. Since $\rho_0^j = \rho_1^j$, the joint distribution of inputs to F_1 and F_2 is identical in $\mathcal{H}_{2k+1}$ and $\mathcal{H}_{2k+2}$. Thus, the joint distribution of outputs from F_1 and F_2 is identical in the two experiments. Thus, the output of P_2 is distributed identically in $\mathcal{H}_{2k+1}$ and $\mathcal{H}_{2k+2}$.

Hybrid $\mathcal{H}_{2k+3}$: This is the ideal world experiment. The simulator S_1^{extOT} extracts (s_0, s_1, r) as above, and sends it to the ideal functionality. The output of P_2 in this case is exactly the output in $\mathcal{H}_{2k+2}$. Thus, this experiment is identical to $\mathcal{H}_{2k+2}$.

Now we handle the case of malicious P_2. Let $\mathcal{A}$ be an adversary controlling P_2, and let $\mathcal{Z}$ be an environment. Let S_{F_1} and S_{F_2} be the simulators for the OTPs for F_1 and F_2 respectively. The ideal-world adversary S_2^{ExtOTM} is defined as follows:

Simulator S_2^{ExtOTM}

1. Receive input bit c from $\mathcal{Z}$ for $\mathcal{A}$. Start internal simulation of $\mathcal{A}$ with given input.

2. For $1 \leq i \leq 2k$, send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}^i)$ to $\mathcal{A}$. Also, run simulators S_{F_1} and S_{F_2} and convey their messages to $\mathcal{A}$.

3. Answer token-queries from $\mathcal{A}$ as follows:

   (a) Let $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}^{i_l}, c^{i_l})$ be a query to one of the M^is. For all but the last such query, answer with a random s^{i_l}, and a random ρ^{i_l}. When $\mathcal{A}$ asks the last query, set $c' = \bigoplus_{j=1}^{2k} c^j$. Send c' to ExtOTM and obtain (s, r). Set $\rho^{i_{2k}} = r \oplus \bigoplus_{l=1}^{2k-1} \rho^{i_l}$. Choose a random $s^{i_{2k}}$, and reply with $(s^{i_{2k}}, \rho^{i_{2k}})$.

   (b) For queries to OTPs for F_1 and F_2, forward them to S_{F_1} and S_{F_2}, and pass their responses back to $\mathcal{A}$.

4. Let $(\hat{s}^1, \ldots, \hat{s}^{2k})$ be S_{F_1}'s query to the ideal functionality for F_1's OTP. If this query occurs before all M^is have been queried, return $\perp$. For $1 \leq j \leq 2k$, check if $\hat{s}^j = s^j$. If not, return $\perp$. Else, return s.

5. Let ρ be S_{F_2}'s query to the ideal functionality for F_2's OTP. If this query occurs before all M^i's have been queried, or if $\rho \neq r$, return $\perp$. Else, return $\rho^1 \circ \ldots \circ \rho^{2k}$ to S_{F_2}.

We proceed to show that $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\text{ExtOTM}}_{S_2^{ExtOTM},\mathcal{Z}}$ are statistically close. Consider the following hybrids:

Hybrid $\mathcal{H}_0$: In this experiment, $\mathcal{Z}$ interacts with simulator S_2^{extOT} only. The simulator receives inputs (s_0, s_1, r) for P_1 on one hand, and bit c for $\mathcal{A}$ on the other. It then internally simulates a real execution of the protocol between P_1 and $\mathcal{A}$, and outputs whatever the simulated $\mathcal{A}$ outputs. Clearly, $\mathcal{H}_0$ is identical to $\mathsf{REAL}_{ExtOTM,\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: S_2^{extOT} runs the simulator S_{F_1} for F_1, and passes its messages to $\mathcal{A}$. When S_{F_1} queries its ideal functionality, S_2^{extOT} runs the correct OTP for F_1 sent by P_1, and returns the output to S_{F_1}. It follows from the security of OTPs that $\mathcal{H}_1$ and $\mathcal{H}_0$ are identical.

Hybrid $\mathcal{H}_2$: If $\mathcal{A}$ queries the OTP for F_1 without querying all the M^is, S_2^{extOT} causes S_{F_1} to abort.

Let M^j be a token not queried by $\mathcal{A}$ before querying F_1. Unless $\mathcal{A}$ guesses one of the outputs of M^j, F_1 will output $\perp$. Thus, $\mathcal{H}_1$ and $\mathcal{H}_2$ are statistically close.

Hybrid $\mathcal{H}_3$: S_2^{extOT} runs the simulator S_{F_2} for F_2, and passes its messages to $\mathcal{A}$. When S_{F_2} queries its ideal functionality, S_2^{extOT} runs the correct OTP for F_2 sent by P_1, and returns the output to S_{F_2}. It follows from the security of OTPs that $\mathcal{H}_2$ and $\mathcal{H}_3$ are identical.

Hybrid $\mathcal{H}_4$: If $\mathcal{A}$ queries F_2 without querying all the M^is, S_2^{extOT} causes S_{F_2} to abort.

Let M^j be a token not queried by $\mathcal{A}$ before querying F_1. Unless $\mathcal{A}$ guesses one the output of M^j, F_1 will output $\perp$. Thus, $\mathcal{H}_1$ and $\mathcal{H}_2$ are statistically close.

Hybrid $\mathcal{H}_5$: For each i, $1 \leq i \leq 2k$, let $\hat{c}^i$ be $\mathcal{A}$'s query to M^i, and let $s^i_{\hat{c}^i} \circ \rho^i$ be its response. Let $(\hat{s}^1, \ldots, \hat{s}^{2k})$ be S_{F_1}'s query to its ideal functionality. S_2^{extOT} checks, for all $1 \leq i \leq 2k$, if $\hat{s}^i = s^i_{\hat{c}^i}$. If not, it returns $\perp$ to S_{F_1}. Else, it returns $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$.

If $s^i_{\hat{c}^i} = \hat{s}^i$ for all i, then $\mathcal{H}_4$ and $\mathcal{H}_5$ are identical. If not, then there exists index j, such that $\hat{s}^j$ is not the value $\mathcal{A}$ received from M^j. Unless $\mathcal{A}$ is able to guess the other output of M^j, F_1 aborts. Thus, $\mathcal{H}_4$ and $\mathcal{H}_5$ are statistically close.

Hybrid $\mathcal{H}_6$: For each i, $1 \leq i \leq 2k$, let $\hat{c}^i$ be $\mathcal{A}$'s query to M^i, and let $s^i_{\hat{c}^i} \circ \rho^i$ be its response. Let ρ be S_{F_2}'s query to its ideal functionality. S_2^{extOT} checks, if $\rho = r$. If not, it returns $\perp$ to S_{F_2}. Else, it returns $\rho^1 \circ \ldots \circ \rho^{2k}$ to S_{F_2}. This is exactly what F_2 does, therefore, $\mathcal{H}_5$ and $\mathcal{H}_6$ are identical.

Hybrid $\mathcal{H}_7$: Let $(\hat{c}^{i_1}, \ldots, \hat{c}^{i_{2k}})$ be $\mathcal{A}$'s queries to the M^is, and let $s^{i_j}_{\hat{c}^{i_j}} \circ \rho^{i_j}$, $1 \leq j \leq 2k-1$ be the first $(2k-1)$ responses. For the final query, instead of running $M^{i_{2k}}$, simulator chooses random k-bit string $t^{i_{2k}}$, and sets $\gamma^{i_{2k}} = r \oplus \bigoplus_{j=1}^{2k-1} \rho^{i_j}$. When S_{F_1} queries its ideal functionality, S_2^{extOT} checks if $\hat{s}^{i_{2k}} = t^{i_{2k}}$ and $\hat{s}^{i_j} = s^{i_j}_{\hat{c}^{i_j}}$ for $i_j \neq i_{2k}$. If all coordinates match, S_2^{extOT} returns $s_{\hat{c}^1 \oplus \ldots \oplus \hat{c}^{2k}}$ to S_{F_1}.

As S_2^{extOT} picks $t^{i_{2k}}$ uniformly at random, $t^{i_{2k}}$ and $s^{i_{2k}}_{c^{i_{2k}}}$ are identically distributed. Also, as $\gamma^{i_{2k}} \oplus \bigoplus_{j=1}^{2k-1} \rho^{i_j} = r$, $\gamma^{i_{2k}}$ and $\rho^{i_{2k}}$ are identically distributed. Thus, the output of $\mathcal{A}$ is identically distributed in experiments $\mathcal{H}_6$ and $\mathcal{H}_7$.

Hybrids $\mathcal{H}_8 \ldots \mathcal{H}_{2k+6}$: In each of these hybrids $\mathcal{H}_{6+l}$, $2 \leq l \leq 2k$, the simulator S_2^{extOT} replaces the outputs of $M^{i_{2k-l+1}}$ with random outputs, as in $\mathcal{H}_7$. By the same argument, these hybrids are identical.

Hybrid $\mathcal{H}_{2k+7}$: This is the ideal world experiment. For the first $2k-1$ queries to M^is, c^{i_j}, S_2^{extOT} responds with random values $s^{i_j} \circ \rho^{i_j}$. For the last query, S_2^{extOT} queries the ideal functionality with bit $\bigoplus_{i=1}^{2k} c^i$, and obtains (s_c, r). Then it sets $\rho^{i_{2k}} = r \oplus \bigoplus_{j=1}^{2k-1} \rho^{i_j}$. This is identical to $\mathcal{H}_{2k+6}$.

□

We are now ready to prove the main feasibility result of this .

Theorem 7. ***(Interactive unconditionally secure computation using stateful tokens.)*** *Let f be a (possibly reactive) polynomial-time computable functionality. Then there exists an efficient, statistically UC-secure interactive protocol which realizes f in the $\mathcal{F}_{wrap}^{single}$-hybrid model.*

Proof We compose three reductions. The protocols of [Kil88, IPS08] realize unconditionally secure two-party (and multi-party) computation of general functionalities using $\mathcal{F}^{\mathrm{OT}}$. A trivial reduction described above reduces $\mathcal{F}^{\mathrm{OT}}$ to ExtOTM. Finally, Claim 6 reduces ExtOTM to $\mathcal{F}_{wrap}^{single}$.

□

While our main focus here is on *feasibility* questions, a couple of remarks about efficiency are in place. First, the protocol Π_{ExtOTM}uses stateful tokens of size poly(k), where k is a statistical security parameter. In the next we will show that the tokens can be further simplified to OTM tokens, each containing a pair of *bits*. Second, the number of stateful tokens employed by the above protocol is proportional to the *computational* complexity of f. This seems unavoidable given the current state of the art in the area of unconditionally secure MPC. However, if one is willing to settle for computational UC-security based on *one-way functions*, Beaver's OT extension technique [Bea96] can be used to reduce the number of tokens to poly(k), independently of the complexity of f. Moreover, all of these poly(k) tokens can be sent at one shot in the beginning of the protocol. We defer a more detailed discussion of these optimizations to the final version.

3.2 The Non-Interactive Setting

In this we restrict the attention to the case of securely evaluating two-party functionalities $f(x, y)$ which take an input x from the sender and an input y from the receiver, and deliver $f(x, y)$ to the receiver. We refer to such functionalities as being *sender-oblivious*. Note that here we consider only *non-reactive* sender-oblivious functionalities, which interact with the sender and the receiver in a single round. The reactive case will be discussed in Section **??**.

Unlike the case of general functionalities, here one can hope to obtain *non-interactive* protocols in which the sender unidirectionally send tokens (possibly along with additional messages[4]) to the receiver.

For sender-oblivious functionalities, the main result of this strengthens the results of Section 3.1 in two ways. First, it shows that a non-interactive protocol can indeed realize such functionalities using stateful tokens. Second, it pushes the simplicity of the tokens to an extreme, relying only on OTM tokens which contain pairs of *bits*.

3.2.1 One-time programs.

Our starting point is the concept of a *one-time program* (OTP) [GKR08]. A one-time program can be viewed in our framework as a non-interactive protocol for $f(x, y)$ which uses only OTM tokens, and whose security only needs to hold for the case of a *semi-honest sender* (and a malicious receiver).[5] The main result of [GKR08] establishes the feasibility of computationally-secure OTPs for any polynomial-time computable f, based on the existence of one-way functions. The construction is based on Yao's garbled circuit technique [Yao86]. Our initial observation is that if f is restricted to the complexity class NC^1, one can replace Yao's construction by an efficient perfectly secure variant (cf. [IK02]). This yields perfectly secure OTPs for NC^1. We now present a general construction of a OTP from any "decomposable randomized encoding" of f. This can be used to derive perfectly secure OTPs for larger classes of functions (including NL) based on randomized encoding techniques from [FKN94, IK02].

The construction uses *randomized encodings* for functions:

Definition 8. ***(Perfect Randomized Encodings*** *[AIK06]) Let $f : \{0,1\}^n \to \{0,1\}^l$ be a function. We say that a function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$ is a perfect randomized encoding of f, if it satisfies the following:*

[4]Since our main focus is on establishing feasibility results, the distinction between the "hardware" part and the "software" part is not important for our purposes.

[5] The original notion of OTP from [GKR08] is syntactically different in that it views f as a function of the receiver's input, where a description of f is given to the sender. This can be captured in our framework by letting $f(x, y)$ be a universal functionality.

- **Correctness** *There exists a deterministic algorithm $D_{\hat{f}}$, called a 'decoder', such that for every input $x \in \{0,1\}^n$,* $\Pr\left[D_{\hat{f}}(\hat{f}(x,\mathcal{U}_m)) \neq f(x) \right] = 0$.

- **Privacy** *There exists a randomized algorithm $S_{\hat{f}}$, called the 'simulator', such that for every $x \in \{0,1\}^n$, $\delta(S_{\hat{f}}(f(x)), \hat{f}(x,\mathcal{U}_m)) = 0$.*

A perfect randomized encoding is called 'efficient' if $\hat{f}$ can be computed in time polynomial in the length of x. A perfect randomized encoding is called 'decomposable' if every output bit of $\hat{f}$ depends upon a single bit of x.

We note that the Correctness and Privacy conditions can be relaxed to obtain computational and statistical randomized encodings.

We will need the following theorem on randomized encodings for $\mathbf{NC}^1$ functions:

Theorem 9. *([Kil88], [IK02]) Let f be an $\mathbf{NC}^1$ function. Then there exists an efficient, perfect decomposable randomized encoding for f.*

Let $f(\cdot,\cdot)$ be any function admitting a decomposable randomized encoding. We now construct an OTP for $f(\cdot,\cdot)$ in the pOTM-hybrid model.

Protocol Π_1. *(One-Time Program for f)*

- ***Input:*** *P_1 has input $x \in \{0,1\}^n$.*

- ***Output:*** *P_2 should receive $f(x,y)$, for $y \in \{0,1\}^n$.*

- ***The Protocol:***

  1. *P_1 constructs a decomposable randomized encoding $\hat{f}(\cdot,\cdot)$ of the function f. Let $\hat{f}_x$ be the restriction of the encoding to x. Choose random string r, and let $\hat{f}_x(y,r) = (f_1(y_1,r),\ldots,f_n(y_n,r))$. Send $(\textsf{create}, \textsf{sid}, P_1, P_2, \textsf{mid}, ((f_1(0,r), f_1(1,r)),\ldots,(f_n(0,r), f_n(1,r))))$ to* pOTM.

  2. *P_2 sends $(\textsf{run}, \textsf{sid}, P_1, P_2, \textsf{mid}, (y_1,\ldots,y_n))$ to* pOTM *and obtains $s := \hat{f}_x(y,r)$. Let $D_{\hat{f}}(\cdot)$ be the decoder for $\hat{f}(\cdot,\cdot)$. Party P_2 outputs $D_{\hat{f}}(s)$.*

□

We now show that the above construction is indeed an OTP for f .

Claim 10. *For any PPT adversary $\mathcal{A}$ corrupting P_2, there exists a PPT Sim_f, such that for every environment $\mathcal{Z}$,*

$$\mathsf{IDEAL}^{\mathcal{F}_f^{OTP}}_{Sim_f,\mathcal{Z}} = \mathsf{REAL}_{\Pi_1,\mathcal{A},Z}.$$

Proof We construct the ideal world simulator Sim_f as follows:

Simulator Sim_f

1. Send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid})$ to $\mathcal{A}$.
2. Receive $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}, (y_1, \ldots, y_n))$ from P_2. Send $y = y_1 \ldots y_n$ to $\mathcal{F}_f^{OTP}$ and obtain $f(x, y)$. Let $S_{\hat{f}}(\cdot)$ be the simulator for the randomized encoding $\hat{f}$. Run $S_{\hat{f}}(f(x, y))$ to obtain $(\rho_1, \ldots, \rho_n)$. Send $(\rho_1, \ldots, \rho_n)$ to $\mathcal{A}$.

It directly follows from Theorem 9 that $\mathsf{IDEAL}^{\mathcal{F}_f^{OTP}}_{Sim_f,\mathcal{Z}} = \mathsf{REAL}_{\Pi_1,\mathcal{A},Z}$.

□

Implementing Parallel-OT tokens by simple OT tokens. The above protocol uses parallel-OT tokens. Now we construct a non-interactive protocol in the OTM-hybrid model that realizes pOTM.

Protocol Π_2 **.**

- ***Input:*** *P_1's input is a tuple of n-bit strings $((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k))$. Party P_2's input is a tuple of bits $(c^1, \ldots, c^k)$.*
- ***Output:*** *P_2 should receive $(s_{c^1}^1, \ldots, s_{c^k}^k)$.*
- ***The Protocol:***

1. *P_1 chooses k random strings r_i, for $1 \le i \le k$. Let $r = r_1 \circ \ldots \circ r_k$. Now P_1 additively shares r into k random shares $\rho_1, \ldots, \rho_k$. For $1 \le i \le k$, party P_1 sends $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i, ((s_0^i \oplus r_i) \circ \rho_i, (s_1^i \oplus r_i) \circ \rho_i))$ to* OTM.

2. *For $1 \le i \le k$, party P_2 sends $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i, c^i)$ to* OTM *and obtains $a_i \circ b_i$. Now party P_2 computes $r' = \sum_{i=1}^{k} b_i$. Let $r'_1, \ldots, r'_k$ be successive n-bit substrings of r'. Party P_2 outputs $(a_1 \oplus r'_1, \ldots, a_k \oplus r'_k)$.*

$\square$

Claim 11. *For any PPT adversary $\mathcal{A}$ corrupting P_2 there exists a PPT S_2^{pOTM}, such that for every environment $\mathcal{Z}$,*

$$\mathsf{IDEAL}^{\mathrm{pOTM}}_{S_2^{pOTM}, \mathcal{Z}} = \mathsf{REAL}_{\Pi_2, \mathcal{A}, \mathcal{Z}}.$$

Proof We define the ideal world adversary S_2^{pOTM} as follows.

Simulator S_2^{pOTM}

1. For $1 \le i \le k$, send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i)$ to $\mathcal{A}$.

2. On input $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_i, c^j)$ from $\mathcal{A}$, do

   (a) if this is not the k^{th} query from $\mathcal{A}$, choose random strings $a_j \in \{0, 1\}^n$ and $b_j \in \{0, 1\}^{kn}$, and return $a_j \circ b_j$ to $\mathcal{A}$.

   (b) if this is the k^{th} query from $\mathcal{A}$, send $(c^1, \ldots, c^k)$ to pOTM and obtain $(s^1_{c^1}, \ldots, s^k_{c^k})$. Choose a random $a_j \in \{0, 1\}^n$. For $1 \le i \le k$, set $r_i = a_i \oplus s^i_{c^i}$. Set $b_j = \sum_{i=1, i \ne j}^{k} b_i + r_1 \circ \ldots \circ r_k$. Send $a_j \circ b_j$.

We proceed to show that for every environment $\mathcal{Z}$, $\mathsf{IDEAL}^{\mathrm{pOTM}}_{S_2^{pOTM}, \mathcal{Z}} = \mathsf{REAL}_{\Pi_2, \mathcal{A}, \mathcal{Z}}$ by considering the following intermediate hybrids. In the following, the symbols $\mathcal{H}_0, \mathcal{H}_1, \ldots$ will be used to denote both the random variable that defines the output of environment $\mathcal{Z}$ in the experiments described, and the experiments themselves.

Hybrid $\mathcal{H}_0$: In this experiment, $\mathcal{Z}$ interacts with S_2^{pOTM} only. Simulator S_2^{pOTM} receives inputs $((s_0^1, s_1^1), \ldots, (s_0^k, s_1^k))$ for P_1, and inputs $(c^1, \ldots, c^k)$ for P_2 from $\mathcal{Z}$. Now S_2^{pOTM} internally simulates a real execution by running P_1 and $\mathcal{A}$ on their respective inputs, and simulating OTM. This is clearly identical to $\mathsf{REAL}_{\Pi_2,\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: This experiment is the same as above, except for $\mathcal{A}$'s final query to (simulated) OTM. Let the final query be $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_j, c^j)$. Let $\{\, a_i \circ b_i \,\}_{\substack{i=1,\ldots k \\ i \neq j}}$ be OTM's responses to $\mathcal{A}$ so far. Instead of sending it to OTM, simulator S_2^{pOTM} answers the last query as follows: choose random string $\hat{a}_j \in \{\, 0, 1 \,\}^n$. Then, for $1 \leq i \leq k$, compute $r'_i = a_i \oplus s^i_{c^i}$, and set $\hat{b}_j = \sum_{i=1, i \neq j}^{k} b_i + r'_1 \circ \ldots \circ r'_k$. Return $\hat{a}_j \circ \hat{b}_j$ to $\mathcal{A}$.

Note that $\mathcal{A}$'s view before the last query is uniformly distributed. Also, $\sum_{i=1, i \neq j}^{k} b_i + \hat{b}_j = r'_1 \circ \ldots \circ r'_k$, and for $1 \leq i \leq k$, $a_i \oplus r'_i = s^i_{c^i}$. Thus, $\mathcal{H}_0$ and $\mathcal{H}_1$ are identical.

Hybrid $\mathcal{H}_2$: This experiment is the same as above, except for the first and last queries by $\mathcal{A}$. Let $(\mathsf{run}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{l_1}, c^{l_1})$ be the first query by $\mathcal{A}$ to OTM. Instead of forwarding this query to OTM, simulator S_2^{pOTM} chooses random strings $\hat{a}_{l_1} \in \{\, 0, 1 \,\}^n$ and $\hat{b}_{l_1} \in \{\, 0, 1 \,\}^{kn}$ and responds with $\hat{a}_{l_1} \circ \hat{b}_{l_1}$. For the last query, S_2^{pOTM} responds as in $\mathcal{H}_1$, using $\hat{a}_{l_1} \circ \hat{b}_{l_1}$ as the response to the first query. All other queries are handled honestly

As before, $\mathcal{A}$'s view before the final query is uniformly distributed. Thus, $\mathcal{H}_1$ and $\mathcal{H}_2$ are identical.

...

Hybrid $\mathcal{H}_k$: This experiment is similar to previous ones, except that S_2^{pOTM} fakes the second-last query also. Thus, in $\mathcal{H}_n$, simulator S_2^{pOTM} answers the first $k-1$ queries by responding with random strings. Then, in the last query, it uses $(s^1_{c^1}, \ldots, s^k_{c^k})$ to correct its previous responses. By the same argument as before, $\mathcal{H}_k$ is identical to $\mathcal{H}_{k-1}$.

Hybrid $\mathcal{H}_{k+1}$: This is the ideal-world experiment. The simulator S_2^{pOTM} proceeds as in $\mathcal{H}_k$ till the last query. At this point, it sends $(c^1, \ldots, c^k)$ to pOTM and obtains $(s^1_{c^1}, \ldots, s^k_{c^k})$.

Then it performs corrections to previous responses as the simulator in $\mathcal{H}_k$.

Note that in $\mathcal{H}_k$, simulator S_2^{pOTM} does not need use P_1's input till after the last query. Thus, $\mathcal{H}_{k+1}$ and $\mathcal{H}_k$ are identical. This completes the proof.

□

A next natural step is to construct unconditionally secure OTPs for any polynomial-time computable function f. However, this result will be subsumed by our main result, which can be proved (in a less self-contained way) without relying on the latter construction.

3.2.2 The Protocol

As in Section 3.1, the main ingredient in our solution is an interactive secure protocol Π for f. To explain the idea it is convenient to first assume that Π is secure in the plain model, without any oracles or setup. In such a case, we could obtain a non-interactive protocol for f which emulates Π by having the sender generate and send a one-time token which computes the sender's next message function for each round of Π. We need to guarantee that the receiver executes the tokens in the correct order, and also let the receiver pass the sender's state information from one token to the other without revealing this information to the sender. This can be done via a standard authentication mechanism: each token i outputs an (unconditionally secure) authenticated encryption of the sender's internal state in the end of Round i, and this information should be supplied by the receiver as an additional input to the token implementing Round $i + 1$. If the authentication fails, the token outputs $\perp$.

The above procedure transforms Π into a non-interactive protocol Π' which uses very complex one-time tokens (for implementing the next message functions of Π). The next idea is that we can break each such complex token into simple OTM tokens by just using a one-time program realization of each complex token. This yields a new non-interactive protocol Π''. The main observation here is that the one-time programs are already secure against a malicious receiver, and any strategy a malicious sender may use in generating badly-formed OTPs corresponds to legitimate strategy for attacking Π' (which in turn corresponds to a legitimate strategy for attacking Π). Note that since the next message function of Π can be assumed wlog to be in NC^1 (possibly breaking each round into multiple mini-rounds), and since unconditionally secure authenticated encryption can also be realized in NC^1, we may assume that each one-time token in Π' realizes an NC^1 function. This allows us to apply the unconditional OTP construction for NC^1 described above.

From the plain model to the OT-hybrid model. Recall that so far we assumed the protocol Π to be secure in the plain model. This rules out unconditional security as well as UC-security, which are our main goals in this section. A natural approach for obtaining unconditional UC-security is to extend the above compiler to protocols in the OT-hybrid model. This introduces a subtle difficulty which was already encountered in Section 3.1: the sender cannot directly implement the OT calls by using OTM tokens, because this would give the receiver an advantage it does not have in Π. Namely, the receiver will be able to defer the invocation of the OTM tokens to the end of the protocol, thereby correlating some of its inputs with partial information obtained from the sender. (The same problem persists even if one applies to Π a standard transformation which guarantees that all OT calls are done in the beginning of the protocol and use random choice bits c_i which are independent of the receiver's inputs.) A natural approach to solve this problem *that does not work in our context*, which was applied in the context of OTPs in [GKR08], is to let the sender secret-share a key between the OTMs which is then used to encrypt all subsequent interaction. However, this gives rise to correlation attacks by a malicious sender, as discussed in Section 3.1. Fortunately, we can use the (non-interactive) ExtOTM protocol from Section 3.1 to realize the approach of [GKR08] while resisting attacks by a malicious sender. The complex tokens required by the ExtOTM protocol can be themselves implemented using one-time programs, thereby eliminating the need for any tokens more complex than simple OTM tokens. We are ready to present our protocol formally now.

Protocol Π'. *(non-interactive protocol for computing $f(x, y)$):*

- ***Input:*** *P_1' has input $x \in \{0, 1\}^n$, and P_2' has input $y \in \{0, 1\}^n$.*

- ***Specified Output:*** *P_2' should receive $f(x, y)$.*

- ***The protocol:***

  1. *P'_1 constructs its first message as follows:*

     (a) *P'_1 uniformly chooses random tape r for P_1. For $1 \leq i \leq l$, P'_1 chooses keys k_1^i for message authentication scheme $MAC(\cdot)$. Also, for $1 \leq i \leq l$, P'_1 chooses random strings k_0^i, where the length of k_0^i is the length of the state output by $g_i^{x,r}(\cdot,\cdot)$.*

     (b) *P'_1 runs P_1 on x and r, and obtains the initial OT values $(r_0^1, r_1^1), \ldots, (r_0^t, r_1^t)$. P'_1 also chooses random strings $\rho_1, \ldots, \rho_t$.*

     (c) *For each $1 \leq i \leq l$, P'_1 constructs an unconditional one-time program for the following functionality G_i:*

        0. *(Only for i=1) Obtain input (m_1, s_0). If $s_0 \neq \rho_1 \circ \ldots \circ \rho_t$, output $\perp$. Else, set $s_0 = \mathsf{null}$ and proceed to step (ii) below.*

        i. *Receive input $(m_i, c_0^{i-1}, c_1^{i-1})$. Check $VF_{k_1^{i-1}}(c_0^{i-1}, c_1^{i-1}) = 1$. If not, output $\perp$. Else, set $s_{i-1} = c_0^{i-1} \oplus k_0^{i-1}$.*

        ii. *Compute $g_i^{x,r}(m_i, s_{i-1})$ to obtain P_1's ith message m, and state s_i.*

        iii. *Output $(m, s_i \oplus k_0^i, MAC_{k_1^i}(s_i \oplus k_0^i))$.*

     (d) *Finally, P'_1's first message comprises of:*

        i. *Tokens for initial OTs: for $1 \leq i \leq t$, P'_1 sends $(P'_1, P'_2, id_i, ((r_0^i, r_1^i), \rho_i))$ to $\mathcal{F}^{ExtOTM}$.*

        ii. *Unconditional OTPs $G_1, \ldots, G_l$.*

  2. ***Output:*** *P'_2 uniformly chooses random tape r' for P_2. Now, P'_2 runs P_2 and executes all initial OTs. Then, for each $1 \leq i \leq l$, P'_2 does the following:*

     (a) *Run P_2 and obtain its message m_i for the ith round.*

     (b) *Run the ith one-time program on input $(m_i, c_0^{i-1}, c_1^{i-1})$ (or (m_i, s_0) if $i = 1$), and obtain (m, c_0^i, c_1^i) as output.*

     (c) *Forward m to P_2.*

     Finally, P'_2 outputs P_2's output.

Theorem 12. *Protocol Π' UC-realizes functionality f in the $(\mathrm{OTM}, \mathcal{F}^{ExtOTM})$-hybrid model.*

Proof

Security against malicious P_1'. Let $\mathcal{A}$ be an adversary corrupting P_1'. We construct an adversary $S_{P_1'}$ such that, for every environment $\mathcal{Z}$, $\mathsf{REAL}_{\Pi',\mathcal{A},\mathcal{Z}} = \mathsf{REAL}_{\Pi,S_{P_1'},\mathcal{Z}}$, i.e., no environment can distinguish between an execution of Π' with adversary $\mathcal{A}$ and an execution of Π with adversary $S_{P_1'}$.

The adversary $S_{P_1'}$ is defined as follows: start internal simulation of adversary $\mathcal{A}$ with input y received from the environment. For each j, $1 \leq j \leq t$, obtain (r_0^j, r_1^j, ρ_j). Use (r_0^j, r_1^j) as the input for the j^{th} OT, and set $s_0 := \rho_1 \circ \ldots \circ \rho_t$. For each i, $1 \leq i \leq l$, obtain OTP for G_i. Now run protocol Π with the external P_2: in round i, receive message m_i from P_2. Run OTP for G_i with input $(m_i, c_0^{i-1}, c_1^{i-1})$ (for $i = 1$, run OTP for G_1 with input (m_1, s_0)). Obtain G_i's response, (m^i, c_0^i, c_i^1). Store (c_0^i, c_1^i) for the next round, and forward m^i externally to P_2.

Note that in both the executions ($(S_{P_1'}, P_2)$ and $(\mathcal{A}, P_2')$), the (joint) distribution of inputs to the one time programs is identical. Thus, the outputs of the OTPs are identically distributed, and the output of P_2' in Π' and P_2 in Π are identically distributed.

Security against malicious P_2'. Let $\mathcal{A}$ be an adversary corrupting P_2', and $\mathcal{Z}$ be any environment. The adversary $S_{P_2'}$ must play the part of P_2 in Π. Adversary $S_{P_2'}$ proceeds as follows: it takes input y from the environment $\mathcal{Z}$ and internally simulates $\mathcal{A}$ on input y. For each $1 \leq i \leq l$, adversary $S_{P_2'}$ invokes the OTP simulator of Claim 10, Sim_{G_i} and sends these simulated OTP $\tilde{G}_i$ to the internal simulation of $\mathcal{A}$. To evaluate $\tilde{G}_i$, $\mathcal{A}$ issues OT queries to $S_{P_2'}$. When $\mathcal{A}$ asks the last input OT query for $\tilde{G}_i$, $\mathcal{A}$'s complete input to $\tilde{G}_i$ is determined, and Sim_{G_i} queries the functionality for the correct output. Now $S_{P_2'}$ externally forwards this to the real P_1, and obtains its response m, which it forwards to Sim_{G_i} as answer to its query.

However, the adversary $\mathcal{A}$ can query the OTs in any order and with arbitrary interleaving between OTs of different OTPs. This is a problem for $S_{P_2'}$. Call the i^{th} OTP *fixed* if all but one of its input OTs have been queried. Thus, when the next input OT request comes, $\mathcal{A}$'s input to this OTP will be fully specified. Now, let $k < i$, and consider the stage where the i^{th} OTP is fixed, while not all OTs for k^{th} OTP have been executed (that is, $\mathcal{A}$'s input to k^{th} OTP is still not fully specified). Now, suppose $\mathcal{A}$ queries the final input OT for i^{th} OTP, thereby fully specifying its input to this OTP. Now, $S_{P_2'}$ can not forward this message to P_1, as P_1 is waiting for a previous response.

To handle this problem, whenever $S_{P_2'}$ detects $\mathcal{A}$ attempting to execute OTP i out of order, it makes it abort; that is, when Sim_{G_i} queries the ideal functionality, $S_{P_2'}$ returns $\perp$. Details follow.

Adversary $S_{P_2'}$

1. Construction phase:
   (a) For $1 \leq i \leq l$, choose keys k_i^0, k_i^1, like P_1'.
   (b) For $1 \leq i \leq l$, run $Sim_{G_i}(\kappa)$ and send its output to $\mathcal{A}$
2. Execution phase: $S_{P_2'}$ handles $\mathcal{A}$'s queries as follows:
   (a) *Initial OTs.* On receiving (P_1', P_2', id_i, c_i) from $\mathcal{A}$ (for $1 \leq i \leq t$), forward c to the external OT, and obtain $r^i_{c_i}$. Choose a random $\hat{\rho}_i$, and return $(r^i_{c_i}, \hat{\rho}_i)$ to $\mathcal{A}$.
   (b) On receiving OT queries from $\mathcal{A}$ for unconditional OTP G_i forward the queries to Sim_{G_i}, and return the response to $\mathcal{A}$.
   (c) For Sim_{G_i}'s query to its ideal functionality, do,
      i. if i^{th} OTP is fixed, and there exists $i' < i$ such that i' is not fixed, return $\perp$.
      ii. else,
         A. if $i = 1$ and all initial OTs have not been queried, then return $\perp$. Else, let σ_1 be $\mathcal{A}$'s input to G_1. Interpret σ_1 as (m_1, s_0). If $s_0 \neq \hat{\rho}_1 \circ \ldots \circ \hat{\rho}_t$, return $\perp$. Else, forward m_1 externally to P_1, and obtain response $\tilde{m}_1$. Choose random state w, and return $(\tilde{m}_1, w \oplus k_0^1, MAC_{k_1^1}(w \oplus k_0^1))$.
         B. else, let σ_i be $\mathcal{A}$'s input to G_i. Interpret σ_i as $(m_i, c_0^{i-1}, c_1^{i-1})$. Verify $VF_{k_1^{i-1}}(c_0^{i-1}, c_1^{i-1}) = 1$. If not, return $\perp$ to Sim_{G_i}. Else, externally forward m_i to P_1, and obtain response $\tilde{m}_i$. Choose random state w, and forward $(\tilde{m}_i, w \oplus k_0^i, MAC_{k_1^i}(w \oplus k_0^i))$ to Sim_{G_i}.

   Finally, output $\mathcal{A}$'s output.

We proceed to show that the random variables $\mathsf{REAL}_{\Pi',\mathcal{A},\mathcal{Z}}$ and $\mathsf{REAL}_{\Pi,S_{P_2'},\mathcal{Z}}$ are statistically close. Consider the following hybrids:

Hybrid $\mathcal{H}_0$: This distribution is the output of the following experiment: the environment $\mathcal{Z}$ interacts with adversary $S_{P_2'}$ only. $S_{P_2'}$ receives input x from $\mathcal{Z}$ for P_1' and input y from $\mathcal{Z}$ for P_2'. Now it internally simulates a real execution of honest P_1' and $\mathcal{A}$ on inputs x and y respectively. Clearly, $\mathcal{H}_0$ is identical to $\mathsf{REAL}_{\Pi',\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: The adversary $S_{P_2'}$ proceeds as above, except for handling the first initial OT. On receiving $(P_1', P_2', id_{i_1}, c_{i_1})$ from $\mathcal{A}$, it sends obtains $(r^{i_1}_{c_{i_1}}, \rho_{i_1})$ from (simulated) $\mathcal{F}^{ExtOTM}$. Then, it chooses a random $\hat{\rho}_{i_1}$, and returns $(r^{i_1}_{c_{i_1}}, \hat{\rho_{i_1}})$ to $\mathcal{A}$.

Note that the distributions of $\hat{\rho}_{i_1}$ in $\mathcal{H}_1$ and ρ_{i_1} in $\mathcal{H}_0$ are identical. Thus, $\mathcal{H}_0$ and $\mathcal{H}_1$ are identical.

Hybrids $\mathcal{H}_2 \ldots \mathcal{H}_t$: In each hybrid $\mathcal{H}_j$, $2 \leq j \leq t$, adversary $S_{P_2'}$ replaces ρ_{i_j} with a random $\hat{\rho}_{i_j}$. As above, all these hybrids are identical.

Hybrid $\mathcal{H}_{t+1}$: Same as above, except adversary $S_{P_2'}$ replaces the l^{th} OTP with a simulated OTP. That is, $S_{P_2'}$ honestly constructs one-time programs for the first $l-1$ next message functions. But for the last one, it runs Sim_{G_l}. When $\mathcal{A}$ completely specifies its input to the last OTP by querying the last input OT for $\tilde{G}_l$, adversary $S_{P_2'}$ replies with the correct response of the l^{th} next message function.

Hybrid $\mathcal{H}_{t+2} \ldots \mathcal{H}_{t+l}$: In each hybrid $\mathcal{H}_{t+j}$, for $2 \leq j \leq l$, $S_{P_2'}$ replaces the $l-j+1$ OTP with $Sim_{G_{l-j+1}}$.

Before proceeding further, we show that the random variables $\mathcal{H}_{t+1}$ and $\mathcal{H}_{t+l}$ are statistically close. For any $1 \leq i < l$ consider any two adjacent hybrids $\mathcal{H}_{t+i}$ and $\mathcal{H}_{t+i+1}$. Observe that the only difference between the two is that in $\mathcal{H}_{t+i}$, the $(l-i)^{th}$ OTP is real, while in

$\mathcal{H}_{t+i+1}$ the $(l-i)^{th}$ OTP is simulated. But by Claim 10, these distributions are statistically close. Thus, $\mathcal{H}_{t+i}$ and $\mathcal{H}_{t+i+1}$ are statistically close, for all $1 \leq i < l$.

Hybrid $\mathcal{H}_{t+l+1}$: Same as $\mathcal{H}_{t+l}$, but now if $\mathcal{A}$ queries the OTP for G_1 before executing all initial OTs, $S_{P'_2}$ returns $\perp$ to Sim_{G_1}.

Note that the OTP for G_1 takes as input $\rho_1 \circ \ldots \circ \rho_t$. Thus, if $\mathcal{A}$ tries to query G_1 OTP before executing all initial OTs, with probability negligibly close to 1, the OTP for G_1 will abort.

Hybrid $\mathcal{H}_{t+l+2}$: Same as before, but now if $\mathcal{A}$ tries to query OTPs out of order, then instead of obtaining the honest output from P_1, $S_{P'_2}$ causes the relevant OTP simulator to output $\perp$.

Note that, in $\mathcal{H}_{t+l+1}$, adversary $\mathcal{A}$ can succeed in out of order querying with probability at most the probability of generating a forged MAC, which is negligible. Thus it follows from the security of the unconditional one-time signature scheme that the two distributions are statistically close.

Hybrid $\mathcal{H}_{t+l+3}$: This experiment is the real execution of Π with $(P_1, S_{P'_2})$, where $S_{P'_2}$ interacts with the real P_1. Observe that in $\mathcal{H}_{t+l+2}$, the adversary does not use P_1's real next message function for construction of any OTPs. Instead, it only uses them to obtain P_1's responses. $\mathcal{H}_{t+l+3}$ is exactly the same except here $S_{P'_2}$ gets P_1's responses directly from P_1 rather than using its next message functions. Thus, the two distributions are identical.

□

Using bit-OT tokens. In all our constructions so far, the size of the output of our tokens is polynomially related to the input size and the security parameter. Ideally, we would only like to use hardware that handles strings of small size. For example, in the case of OT tokens, we would like to use only bit OT tokens. To this end, one can use a known *perfectly secure* reduction from string OT to bit OT [BCS96]. This reduction reduces OT on

ℓ-bit strings to $O(\ell)$ parallel instances of bit OT. While in our case we need the reduction to be UC-secure against a receiver who may invoke the bit OTs in an arbitrary adaptive fashion, the reduction from [BCS96] can indeed be shown to satisfy this stronger notion of security. This is a natural consequence of the perfect security of the reduction and an efficient conditional sampling property. In the appendix, we briefly sketch an alternative construction and proof using a coding argument. Further details will be provided in the final version. Combining this final reduction with the previous results, we get an unconditionally-secure, non-interactive, UC-secure protocol for a circuit C which uses $O(|C| \cdot \text{poly}(\kappa))$ bit OT tokens.

This yields the following main result of this section:

Theorem 13. ***(Non-interactive unconditionally secure computation using bit-OTM tokens.)*** *Let $f(x, y)$ be a non-reactive, sender-oblivious, polynomial-time computable two-party functionality. Then there exists an efficient, statistically UC-secure non-interactive protocol which realizes f in the $\mathcal{F}_{wrap}^{single}$-hybrid model in which the sender only sends bit-OTM tokens to the receiver.*

CHAPTER 4

Two-Party Computation with Stateless Tokens

In this chapter, we again address the question of achieving interactive two-party computation protocols, but asking the following questions: (1) Can we achieve interactive two-party computation protocols without requiring that the number of tokens increase with the complexity of the function being computed, as was the case in the previous section, and (2) Can we achieve two-party computation protocols using *stateless* tokens? We show how to positively answer both questions: We use stateless tokens, whose complexity is polynomial in the security parameter, to implement the OT functionality. We assume only the existence of one-way functions. Since (as discussed earlier), secure protocols for any two-party task exist given OT, this suffices to achieve the claimed result. Our construction for the OT functionality (and thus for general two-party computation) is UC secure.

Before turning to our protocols, we make a few observations about stateless tokens to set the stage. First, we observe that with stateless tokens, it is always possible to have protocols where tokens are exchanged *only at the start of the protocol*. This is simply because each party can create a "universal" token that takes as input a pair (c, x), where c is a (symmetric authenticated/CCA-secure) encryption[1] of a machine M, and outputs $M(x)$. Then, later in the protocol, instead of sending a new token T, a party only has to send the encryption of the code of the token, and the other party can make use of that encrypted code and the universal token to emulate having the token T. The proof of security and correctness of this construction is straightforward, and omitted for the sake for brevity.

Dealing with dishonestly created *stateful* tokens. The above discussion, however, assumes that dishonest players also only create stateless tokens. If that is not the case, then

[1] An "encrypt-then-MAC" scheme would suffice here.

re-using a dishonestly created token may cause problems with security. If we allow dishonest players to create stateful tokens, then a simple solution is to repeat the above construction and send separate universal tokens for each future *use* of any token by the other player, and honest players are instructed to only use each token once. Since this forces all tokens to be used in a stateless manner, this simple fix is easily shown to be correct and secure; however, it may lead to a large number of tokens being exchanged. To deal with this, as was discussed in the previous section, we observe that by Beaver's OT extension result [Bea96] (which requires only one-way functions), it suffices to implement $O(k)$ OT's, where k is the security parameter, in order to implement any polynomial number of OT's. Thus, it suffices to exchange only a linear number of tokens even in the setting where dishonest players may create stateful tokens.

Convention for intuitive protocol descriptions. In light of the previous discussions, in our protocol descriptions, in order to be as intuitive as possible, we describe tokens as being created at various points during the protocol. However, as noted above, our protocols can be immediately transformed into ones where a bounded number of tokens (or in the model where statelessness is guaranteed, only one token each) are exchanged in an initial setup phase.

4.1 Protocol Intuition

We now discuss the intuition behind our protocol for realizing OT using stateless tokens; due to the complexity of the protocol, we do not present the intuition for the entire protocol all at once, but rather build up intuition for the different components of the protocol and why they are needed, one component at a time. For this intuition, we will assume that the sender holds two *random* strings s_0 and s_1, and the receiver holds a choice bit b. Note that OT of random strings is equivalent to OT for chosen strings [BG89].

The Basic Idea. Note that, since stateless tokens can be re-used by malicious players, if we naively tried to create a token that output s_b on input the receiver's choice bit b, the receiver could re-use it to discover both s_0 and s_1. A simple idea to prevent this reuse would be the following protocol, which is our starting point:

1. Receiver sends a commitment $c = \mathsf{com}(b; r)$ to its choice bit b.
2. Sender sends a token, that on input (b, r), checks if this is a valid decommitment of c, and if so, outputs s_b.
3. Receiver feeds (b, r) to the token it received, and obtains $w = s_b$

Handling a Malicious Receiver. Similar to the problem discussed in the previous section, there is a problem that the receiver may choose not to use the token sent by the sender until the end of the protocol (or even later!). In our context, this can be dealt with easily. We can have the sender commit to a random string π at the start of the protocol, and require that the sender's token must, in addition to outputting s_b, also output a valid decommitment to π. We then add a last step where the receiver must report π to the sender. Only upon receipt of the correct π value does the sender consider the protocol complete.

Handling a Malicious Sender. While this protocol seems intuitive, we note that it is actually *insecure* for a fairly subtle reason. A dishonest sender could send a token that on input (b, r), simply outputs (b, r) (as a string). This means that at the end of the protocol, the dishonest sender can output a specific commitment c, such that the receiver's output is a decommitment of c showing that it was a commitment to the receiver's choice bit b. It is easy to see that this is impossible in the ideal world, where the sender can only call an ideal OT functionality.

To address the issue above, we need a way to prevent the sender from creating a token that can adaptively decide what string it will output. This has to be done in a way to enable our simulator to extract the inputs of the malicious sender. Thinking about it in a different way, we want the sender to "prove knowledge" of two strings before he sends his token. We can accomplish this by adding the following preamble to the protocol above:

1. Receiver chooses a pseudo-random function (PRF) $f_\gamma : \{0,1\}^{5k} \to \{0,1\}^k$, and then sends a token that on input $x \in \{0,1\}^{5k}$, outputs $f_\gamma(x)$.

2. Sender picks two strings $x_0, x_1 \in \{0,1\}^{5k}$ at random, and feeds them (one-at-a-time) to the token it received, and obtains y_0 and y_1. The sender sends (y_0, y_1) to the receiver.

3. Sender and receiver execute the original protocol above with x_0 and x_1 in place of s_0 and s_1. The receiver checks to see if the string w that it obtains from the sender's token satisfies $f_\gamma(w) = y_b$, and aborts if not.

The crucial feature of the protocol above is that a dishonest sender is effectively committed to two values x_0 and x_1 after the second step (and in fact the simulator can use the PRF token to extract these values), such that later on it must output x_b on input b, or abort.

Note that a dishonest receiver may learn k bits of useful information about x_0 and x_1 each from its token, but this can be easily eliminated later using the Leftover Hash Lemma (or any strong extractor).

Preventing correlated aborts. A final significant subtle obstacle remains, however. A dishonest sender can still send a token that causes an abort to be correlated with the receiver's input, *e.g.* it could choose whether or not to abort based on the inputs chosen by the receiver[2].

To prevent a dishonest sender from correlating the probability of abort with the receiver's choice, the input b of the receiver is additively shared into bits $b_1, \dots, b_k$ such that $b_1 + b_2 + \cdots + b_k = b$. The sender, on the other hand, chooses strings $z_1, \dots, z_k$ and r uniformly at random from $\{0,1\}^{5k}$. Then the sender and receiver invoke k parallel copies of the above protocol (which we call the *Quasi-OT* protocol), where for the ith execution, the sender's inputs are $(z_i, z_i + r)$, and the receiver's input is b_i. Note that at the end of the protocol, the receiver either holds $\sum z_i$ if $b = 0$, or $r + \sum z_i$ if $b = 1$.

[2]At first glance, this may not seem like a problem, since we can treat an abort as a special output string $\perp$, which the sender could have anyway provided as one his inputs to the OT. But the adaptive decision of whether or not to abort is actually a problematic additional "axis" of control that the sender has in addition to the allowed choice of strings depending on the receiver's bit. We now elaborate with an example:

A concrete "problem case" is to consider a commitment algorithm in which the first bit c_1 of the commitment to a bit b is set equal to $r \oplus b$, where r is a randomly chosen bit. Now, a dishonest sender can send a token such that when it is fed the decommitment information, it decides to abort iff $r = 1$. Let $r' = 1$ iff the honest receiver sees the token abort and thus aborts. In real life executions of the protocol, we will always have the invariant that $c_1 = r' \oplus b$. However the natural simulator for this protocol, in which the simulator (which does not know b), chooses a commitment to a random b', would lead to ideal world executions in which $c_1 \neq r' \oplus b$ with probability 1/2.

Intuitively speaking, this reduction (variants of which were previously used by, e.g. [Kil90, LP07]) forces the dishonest sender to make one of two bad choices: If each token that it sends aborts too often, then with overwhelming probability at least one token will abort and therefore the entire protocol will abort. On the other hand, if few of the sender's tokens abort, then the simulator will be able to perfectly simulate the probability of abort, since the bits b_i are $(k-1)$-wise independent (and therefore all but one of the Quasi-OT protocols can be perfectly simulated from the receiver's perspective). We make the receiver commit to its bits b_i using a statistically hiding commitment scheme (which can be constructed from one-way functions [HR07]) to make this probabilistic argument go through.

Now we are ready to present our protocol formally.

4.1.1 Preliminaries

Statistically Hiding Commitment Schemes We will use the statistically hiding commitment scheme of [HR07]. The receiver's transcript of commitment to bit b when the sender uses randomness r will be denoted by $\mathsf{scom}(b, r)$. The decommitment phase consists of the sender simply sending its randomness r along with bit b, and the receiver verifies if (r, b) is consistent with the transcript $\mathsf{scom}(b, r)$.

Definition 14. *(Pairwise Independent Hash Functions ([CW79])) Let $\mathcal{H}$ be a family of functions mapping strings of length $l(n)$ to strings of length $m(n)$. Then, $\mathcal{H}$ is an **efficient family of pairwise independent hash functions** if the following hold:*

Samplable. *$\mathcal{H}$ is polynomially samplable in n.*

Efficient. *There exists a polynomial time algorithm that given $x \in \{0,1\}^{l(n)}$ and a description of $h \in \mathcal{H}$ outputs $h(x)$.*

Pairwise Independence. *For every distinct $x_1, x_2 \in \{0,1\}^{l(n)}$, and every $y_1, y_2 \in \{0,1\}^{m(n)}$, we have:*

$$\Pr_{h \leftarrow \mathcal{H}}[\, h(x_1) = y_1 \wedge h(x_2) = y_2 \,] = 2^{-2m(n)}.$$

It is known ([CW79]) that there exists an efficient family of pairwise independent hash functions for every l and m whose element description size is $\mathcal{O}(\mathsf{max}(l(n), m(n)))$.

Let X be a random source with min-entropy k.

Theorem 15. *(Leftover Hash Lemma ([HILL99])) If the family $\mathcal{H}$ of hash functions $h : \{0,1\}^n \to \{0,1\}^l$ is pairwise independent, where $l = k - 2\log(1/\epsilon) - \mathcal{O}(1)$, then $Ext(X, h) := (h, h(X))$ is a strong $(k, \epsilon/2)$-extractor.*

4.1.2 The Protocol

Let k be the security parameter. Let $\mathcal{H}$ be an efficient family of pairwise hash functions mapping strings of length $5k$ to k. Let $\mathcal{F}$ be a family of pseudo-random functions mapping strings of length $5k$ to k.

Protocol. *(Quasi OT.)*

- ***Input:*** *P_1 has two strings $(s_0, s_1) \in \{0,1\}^k$, P_2 has a selection bit b.*
- ***Common input:*** *An index j.*
- ***Protocol:***
  1. *P_2 chooses a random PRF key γ for family $\mathcal{F}$, and sends $(\mathsf{create}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1}, M_1)$, where M_1 implements the following functionality:*
     - *On input string $x \in \{0,1\}^{5k}$, output $f_\gamma(x)$.*

     P_2 also sends to P_1 a randomly chosen string r, which serves as the first message of the (statistically binding) commitment scheme com.
  2. *P_1 chooses two strings x_0 and x_1 uniformly from $\{0,1\}^{5k}$, and a random string $\pi \in \{0,1\}^k$. P_1 sends $(\mathsf{run}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1}, x_i)$ to obtain y_i, for $i \in \{0,1\}$. Now P_2 chooses a random hash function $h \in \mathcal{H}$, and sends $(y_0, y_1, h, \alpha = \mathsf{com}(\pi))$ to P_2.*

3. *Now P_2 commits to its input bit b using the commitment scheme* scom. *That is, P_2 and P_1 run the statistically hiding commitment protocol* scom*, with P_2 acting as the sender in the commitment protocol with input bit b. Let $\hat{\alpha}$ be P_1's transcript (view) of the commitment phase, and let $\hat{r}$ be the randomness used by P_2 in the commitment protocol.*

4. *If the commitment phase succeeds, P_1 sends $(\textit{create}, \textit{sid}, P_1, P_2, \textit{mid}_{j,2}, M_2)$, where M_2 implements the following functionality:*

   - *Obtain randomness $\hat{r}$ and bit b, and verify that this is a valid decommitment (with respect to commitment transcript $\hat{\alpha}$). If so, output x_b and $\beta = \mathsf{open}(\alpha)$. Else, output $\perp$.*

5. *P_2 sends $(\textit{run}, \textit{sid}, P_1, P_2, \textit{mid}_{j,2}, (\hat{r}, b))$ and obtains x_b and β. It then checks if $f_\gamma(x_b) = y_b$. If not, it aborts. Then it checks if β is a valid opening of α. If not, it aborts. Else, let π' be the revealed string. P_2 sends π' to P_1.*

6. *P_1 receives string π' from P_2, and checks if $\pi = \pi'$. If not, it aborts. Else, it sends $(s'_0 = s_0 \oplus h(x_0), s'_1 = s_1 \oplus h(x_1))$.*

7. *P_2 receives (s'_0, s'_1), and outputs $s'_b \oplus h(x_b)$.*

As mentioned before, this protocol does not realize the OT functionality, as a malicious sender can selectively abort based on receiver's input. Now we present a protocol that uses QuasiOT as a subroutine and realizes OT in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model.

Protocol. *(OT in $\mathcal{F}_{wrap}^{stateless}$-hybrid model.)*

- ***Input:*** *P_1 has two strings $(s_0, s_1) \in \{0,1\}^k \times \{0,1\}^k$, P_2 has selection bit b.*

- ***Output:*** *P_2 outputs s_b.*

- ***Protocol:***

  1. *P_1 chooses $z_1, \ldots, z_k, r \in \{0,1\}^k$ uniformly at random. P_2 chooses $k-1$ random bits $b_1, \ldots, b_{k-1}$, and sets b_k such that $b = \bigoplus_{j=1}^{k} b_j$. Now P_1 and P_2 execute in*

parallel, k copies of QuasiOT. For $1 \leq j \leq k$, the inputs to the j^{th} copy of QuasiOT are $(z_j, z_j \oplus r)$ and b_j.

2. *If all k copies of QuasiOT finish without aborts, then P_1 sends to P_2 the pair $(s'_0 = s_0 \oplus \bigoplus_{j=1}^{k} z_j, s'_1 = s_1 \oplus \bigoplus_{j=1}^{k} z_j \oplus r)$.*
3. *For $1 \leq j \leq k$, let a_j be the string received by P_2 at the end of the j^{th} invocation of QuasiOT. Then, P_2 outputs $s'_b \oplus \bigoplus_{j=1}^{k} a_j$.*

4.1.3 Security Proof

Theorem 16. *Protocol 4.1.2 UC-realizes OT in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model.*

Proof We first consider the case of malicious sender. Let $\mathcal{A}$ be an adversary corrupting P_1, and let $\mathcal{Z}$ be any environment. For each j, $1 \leq j \leq k$, we first present a sub-routine $\hat{S}_{1,j}$ that will be used by the ideal-world simulator S_1.

Subroutine $\hat{S}_{1,j}$

1. Send $(\mathsf{create}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1})$ to $\mathcal{A}$. Also send a random string r.

2. For each query x to $\mathsf{mid}_{j,1}$, $\hat{S}_{1,j}$ checks if $\mathcal{A}$ queried the $\mathsf{mid}_{j,1}$ on x before. If it did, return the previous response. Else, return a randomly chosen k-bit string y. The simulator keeps a list of $\mathcal{A}$'s queries to $\mathsf{mid}_{j,1}$ and its responses.

3. When $\hat{S}_{1,j}$ receives (y_0, y_1, h, α) from $\mathcal{A}$, it checks the list of responses, and obtains the strings x_0 and x_1 to which it responded with y_0 and y_1 respectively. If for any y_i, $i \in \{0, 1\}$, no such x_i exists, set $x_i = \perp$.

4. $\hat{S}_{1,j}$ picks a random bit b, and runs the commitment protocol scom with $\mathcal{A}$, with b as its input.

5. When $\hat{S}_{1,j}$ receives $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{j,2}, M_{j,2})$, it honestly runs the decommitment phase of scom, and obtains output $(\hat{x}_b, \beta)$. If $\hat{x}_b \neq x_b$, it aborts. If β is not the correct opening of α, it aborts. Else, let π be the string revealed in the opening. $\hat{S}_{1,j}$ returns π to $\mathcal{A}$.

6. When $\hat{S}_{1,j}$ obtains $(s'_{j,0}, s'_{j,1})$ from $\mathcal{A}$, it returns $(s_{j,0} = s'_{j,0} \oplus h(x_0), s_{j,1} = s'_{j,1} \oplus h(x_1))$ to S_1.

Now we describe the the 'outer' simulator S_1.

Simulator S_1

1. Invoke adversary $\mathcal{A}$, and pass messages from environment to P_1 or $\mathcal{A}$, to $\mathcal{A}$ and vice versa.

2. For each j, $1 \leq j \leq k$, run $\hat{S}_{1,j}$, and pass messages between the subroutine and $\mathcal{A}$. If no subroutine $\hat{S}_{1,j}$ aborts, for each j, obtain pair $(s_{j,0}, s_{j,1})$.

3. Let (s'_0, s'_1) be the final message from $\mathcal{A}$. The simulator S_1 picks two random bit-vectors $(b_1, \ldots, b_k)$ and $(b'_1, \ldots, b'_k)$, such that $\bigoplus_{j=1}^{k} b_j = 0$ and $\bigoplus_{j=1}^{k} b'_j = 1$. Then it sets $s_0 = s'_0 \oplus \bigoplus_{j=1}^{k} s_{j,b_j}$ and $s_1 = s'_1 \oplus \bigoplus_{j=1}^{k} s_{j,b'_j}$. The simulator sends (s_0, s_1) to the ideal OT functionality.

We prove indistinguishability of real and ideal worlds via the following series of hybrids. We denote the k invocations of the QuasiOT subroutine by $QOT_1, \ldots, QOT_k$.

Hybrid $\mathcal{H}_0$: This is the real execution.

Hybrid $\mathcal{H}_{1,0}$: Same as above, except we modify the behaviour of QOT_1 as follows: instead of answering $\mathcal{A}$'s queries by running $M_{1,1}$, for each query x, we respond with a randomly chosen k-bit string. We record the queries and our responses in a list. When $\mathcal{A}$ asks query x, we first check if $\mathcal{A}$ asked x before. If this is the case, we respond with the same string as before.

Any environment that can distinguish between $\mathcal{H}_{1,0}$ and $\mathcal{H}_0$ can distinguish the PRF from a random function. Thus, as the PRF key is never revealed, by the security of the PRF, hybrids $\mathcal{H}_{1,0}$ and $\mathcal{H}_0$ are indistinguishable.

Hybrid $\mathcal{H}_{1,1}$: Same as above, except now, as we record the responses in the list, we ensure that no two of $\mathcal{A}$'s queries have the same response. If there is a collision, $\mathcal{H}_{1,1}$ aborts.

Note that the probability of abort in $\mathcal{H}_{1,1}$ is at most the probability of finding a collision in a randomly chosen function. Thus, $\mathcal{H}_{1,0}$ and $\mathcal{H}_{1,1}$ are statistically close.

Hybrid $\mathcal{H}_{1,2}$: This experiment is same as above, except the following: let (y_0, y_1, h, α) be $\mathcal{A}$'s first message to P_2. We check the list of responses to find strings x_0 and x_1 such that y_0 and y_1 were the responses to x_0 and x_1 respectively. Note that conditioned on the event that $\mathcal{H}_{1,1}$ does not abort, each y can be a response to at most a single $\mathcal{A}$ query x. If for $i \in \{0, 1\}$, y_i is not the response to any query, set $x_i = \perp$. Now, let $(\hat{x}_{b_1}, \beta)$ be $M_{1,2}$'s output. If $\hat{x}_{b_1} \neq x_{b_1}$, hybrid $\mathcal{H}_{1,2}$ outputs $\perp$. Else, let $(s'_{1,0}, s'_{1,1})$ be $\mathcal{A}$'s final message. QOT_1 returns $(s_{1,0} = s'_{1,0} \oplus h(x_0), s_{1,1} = s'_{1,1} \oplus h(x_1))$ to S_1, which forwards s_{1,b_1} to the simulated P_2.

If $x_{b_1} \neq \perp$, the the probability that the images (under a random function) of $\hat{x}_{b_1}$ and x_{b_1} would be the same, is negligible. If $x_{b_1} = \perp$, then the probability that $\mathcal{A}$ can guess the image (under a random function) of $\hat{x}_{b_1}$ is negligible. Thus, $\mathcal{H}_{1,1}$ and $\mathcal{H}_{1,2}$ are statistically close.

Hybrids $\mathcal{H}_{2,0}, \ldots, \mathcal{H}_{k,2}$: For $2 \leq j \leq k$ and $j' \in \{0, 1, 2\}$, hybrid $\mathcal{H}_{j,j'}$ modifies the behaviour of QOT_j in the same way as $\mathcal{H}_{1,j'}$ modifies the behaviour of QOT_1. By the same arguments as above, it follows that these hybrids are indistinguishable.

Hybrid $\mathcal{H}_{k+1}$: Same as above, except instead of using $\vec{v} = (b_1, \ldots, b_k)$ as the inputs to QOT_js, where $\bigoplus_{j=1}^{k} b_j = b$, we pick a random vector of bits $\vec{v}'$, and use that as inputs to the QOT_js. Let (s'_0, s'_1) be the final message from $\mathcal{A}$. Now we randomly pick two k-bit vectors $\vec{v}_0 = (b_1, \ldots, b_k)$ and $\vec{v}_1 = (b'_1, \ldots, b'_k)$ such that $\bigoplus_{j=1}^{k} b_j = 0$ and $\bigoplus_{j=1}^{k} b'_j = 1$. For each j, let $(s_{j,0}, s_{j,1})$ be the pairs of strings obtained from QOT_js. Set $s_0 = s'_0 \oplus \bigoplus_{j=1}^{k} s_{j,b_j}$ and $s_1 = s'_1 \oplus \bigoplus_{j=1}^{k} s_{j.b'_j}$. Finally, output s_b as P_2's output.

Note that com is a statistically binding commitment scheme. Thus, with probability negligibly close to 1, α has a single opening. Next, observe that the only difference between $\mathcal{H}_{k,2}$ and $\mathcal{H}_{k+1}$ is in the inputs to the OT boxes (that is, inputs to the tokens $M_{j,2}$). Also note that in $\mathcal{H}_{k,2}$, each token $M_{j,2}$, outputs either the correct string x_{j,b_j}, or $\perp$. Thus, the only difference in the two hybrids is the probability of abort. We now show that the probabilities of abort are negligibly close to each other.

Claim 17. *Let p_1 be the probability of abort in $\mathcal{H}_{k,2}$, and p_2 be the probability of abort in*

$\mathcal{H}_{k+1}$. *Then,* $|p_1 - p_2| \leq 2^{-k+1}$.

Proof For simplicity, we assume that the commitment scheme scom is *perfectly* hiding. In the end, we point out how to extend the proof to the case of statistical hiding schemes.

We will condition the probability of abort on $\mathcal{A}$'s transcripts of the commitments. Let $\vec{v} = (b_1, \ldots, b_k)$ be a bit vector, and let $\vec{r} = (r_1, \ldots, r_k)$ be a vector of random strings. By $\mathsf{scom}(\vec{v}, \vec{r}) = (\hat{\alpha}_1, \ldots, \hat{\alpha}_k)$ we mean a vector of transcripts, where for $1 \leq j \leq k$, $\hat{\alpha}_j$ is the transcript of the receiver in the commitment scheme scom when the sender commits bit b_j using randomness r_j (the randomness of the receiver is implicit).

Let $\vec{c} = (\hat{\alpha}_1, \ldots, \hat{\alpha}_k)$ be a transcript vector. First, note that, because of the perfect hiding property of the commitment scheme scom, the probability of occurrence of $\vec{c}$ in $\mathcal{H}_{k,2}$ and $\mathcal{H}_{k+1}$ is exactly the same. That is, the number of randomness strings $\vec{r}$ that lead to transcript $\vec{c}$ is the same in the two hybrids. For $\vec{c}$, fix a randomness vector $\vec{r}$ that can lead to $\vec{c}$. For $\vec{c}$ and $\vec{r}$, we analyze the behaviour of the tokens $M_{j,2}$ in the following two experiments corresponding to $\mathcal{H}_{k,2}$ and $\mathcal{H}_{k+1}$ respectively: in the first, $\vec{v}$ is chosen so that $\bigoplus_{j=1}^{k} b_j = b$, and in the other, $\vec{v}$ is randomly chosen. Note that fixing $\vec{c}$ and $\vec{r}$ fixes the output of token $M_{j,2}$: on on input b_j, it either outputs x_{j,b_j} or $\perp$. Consider the following two cases:

Case 1. There exists j, $1 \leq j \leq k$ such that $M_{j,2}$ aborts neither on 0 nor on 1. In this case, as the inputs to the $M_{j',2}$s are $(k-1)$-wise independent, the probability of abort is identical in the two cases.

Case 2. For all j, $M_{j,2}$ aborts on either 0 or 1. Then, the probability that $\mathcal{H}_{k,2}$ aborts is at least $1 - 2^{-k+1}$. Thus, in this case, the difference in the probability that $\mathcal{H}_{k,2}$ aborts and the probability that $\mathcal{H}_{k+1}$ aborts is at most 2^{-k+1}.

Thus, the two hybrids are statistically close.

In the case of statistical hiding instead of perfect hiding, the probability that a transcript vector $\vec{c}$ occurs in the two hybrids are not the same, but negligibly close. Thus, we must discard some randomness vectors $\vec{r}$ from the analysis, but this changes the probabilities only be a negligible amount.

$\square$

Hybrid $\mathcal{H}_{k+2}$: This is the ideal world. Note that we only use P_2's selection bit b in the final step to determine its output. As the ideal world P_2 is honest, it queries the ideal functionality with bit b and obtains string s_b. Thus, $\mathcal{H}_{k+2}$ and $\mathcal{H}_{k+1}$ are identical.

Next, we handle the case of a malicious receiver. Let $\mathcal{A}$ be an adversary corrupting P_2, and let $\mathcal{Z}$ be any environment. We first present $\hat{S}_{2,j}$ which will be used as a sub-routine by simulator S_2.

Subroutine $\hat{S}_{2,j}$

1. Receive as input from S_2 a pair of strings $(s_{j,0}, s_{j,1})$. Receive $(\mathsf{create}, \mathsf{sid}, P_2, P_1, \mathsf{mid}_{j,1}, M_{j,1})$ and random string r from $\mathcal{A}$. Run $M_{j,1}$ on randomly chosen $5k$-bit strings $x_{j,0}$ and $x_{j,1}$ and obtain $y_{j,0}$ and $y_{j,1}$. Choose random $\pi_j \in \{0,1\}^k$ and a random hash function $h_j \in \mathcal{H}$, and return $(y_{j,0}, y_{j,1}, h_j, \alpha_j = \mathsf{com}(\pi_j))$ to $\mathcal{A}$.

2. Run the statistical hiding commitment protocol with $\mathcal{A}$. Let $\hat{\alpha}_j$ be the receiver's transcript.

3. Send $(\mathsf{create}, \mathsf{sid}, P_1, P_2, \mathsf{mid}_{j,2})$ to $\mathcal{A}$. If $\mathcal{A}$ correctly decommits, then let b_j be the revealed bit. Send $(x_{j,b_j}, \beta_j = \mathsf{open}(\alpha_j))$ to $\mathcal{A}$. If $\mathcal{A}$ ever decommits to both 0 and 1, output $\mathsf{binding\ abort}$ and abort.

4. Receive π' from $\mathcal{A}$. If $\pi' \neq \pi$, abort. If $\mathcal{A}$ returns the correct π without running $M_{j,2}$, output $\mathsf{hiding\ abort}$ and abort. Else, set $s'_{j,0} = s_{j,0} \oplus h_j(x_{j,0})$ and $s'_{j,1} = s_{j,1} \oplus h_j(x_{j,1})$. Send $(s'_{j,0}, s'_{,1})$ to $\mathcal{A}$, and return b_j to S_2.

Now we present the ideal-world simulator S_2.

Simulator S_2

1. Choose $z_1, \ldots, z_k, r \in \{0, 1\}^k$ uniformly at random. For each $1 \leq j \leq k$, run $\hat{S}_{2,j}$ with input $(z_j, z_j \oplus r)$. If any subroutine aborts, abort. Else, let b_j be the values returned by the $\hat{S}_{2,j}$s.

2. Send $b = \bigoplus_{j=1}^{k} b_j$ to the ideal OT functionality, and obtain s_b. Choose random k-bit string $s_{1\oplus b}$. Set $s'_b = s_b \oplus \bigoplus_{j=1}^{k}(z_j \oplus b_j \cdot r)$ (where $b_j \cdot r$ is r if $b_j = 1$, else 0) and $s'_{1\oplus b} = s_{1\oplus b}$. Send (s'_0, s'_1) to $\mathcal{A}$.

We show indistinguishability of the real and ideal worlds via the following hybrids.

Hybrid $\mathcal{H}_0$: This is the real world execution.

Hybrid $\mathcal{H}_1$: Same as above, except we modify the behaviour of the QuasiOT protocols in the following way: if for any j, $\mathcal{A}$ outputs the correct value π_j without running $M_{j,2}$, then output $\mathsf{hiding\ abort}$ and abort.

We now show that the probability that $\mathcal{H}_1$ outputs $\mathsf{hiding\ abort}$ is negligible.

Claim 18. *Let ϵ be the probability that $\mathcal{H}_1$ outputs $\mathsf{hiding\ abort}$. Then, ϵ is negligible in k.*

Proof We construct a non-uniform adversary $\mathcal{A}_{\mathsf{com}}$ that breaks the hiding property of commitment scheme com with probability ϵ. To show this, the adversary $\mathcal{A}_{\mathsf{com}}$ must succeed with probability ϵ in the following game: $\mathcal{A}_{\mathsf{com}}$ interacts with an external party, called the challenger, and acts as the receiver in commitment scheme com. It sends a random string r as the first message, and receives a commitment $\tilde{\alpha}$ to a random k bit string π. The adversary succeeds if it outputs π.

Let j be such that $\mathcal{A}$ outputs π_j without running $M_{j,2}$ with probability at least ϵ. The adversary $\mathcal{A}_{\mathsf{com}}$ works as follows: it sends a random r to the challenger, and receives a commitment $\tilde{\alpha} = \mathsf{com}(\pi)$. Now $\mathcal{A}_{\mathsf{com}}$ sets up an execution of $\mathcal{A}$ and honest P_1 with environment $\mathcal{Z}$. For the j^{th} copy of the QuasiOT subroutine, it uses $\tilde{\alpha}$ as the commitment in P_1's first message to $\mathcal{A}$. Thereafter, $\mathcal{A}_{\mathsf{com}}$ proceeds with the rest of the execution without modification. If $\mathcal{A}$ queries $M_{j,2}$ correctly, then $\mathcal{A}_{\mathsf{com}}$ aborts. Otherwise, let π' be $\mathcal{A}$'s message to P_1 in step 6 of the j^{th} QuasiOT subroutine. Adversary $\mathcal{A}_{\mathsf{com}}$ outputs π' as its guess of the string committed in $\tilde{\alpha}$.

By the hypothesis, with probability ϵ, adversary $\mathcal{A}$ outputs the correct stri ng π in step 6 of the j^{th} QuasiOT. Thus, if the commitment scheme com is computationally hiding, ϵ must be negligible.

$\square$

Hybrid $\mathcal{H}_2$: This is the same as above, except if for any j, adversary $\mathcal{A}$ provides two different openings to $M_{j,2}$, then $\mathcal{H}_2$ outputs $\mathsf{binding\ abort}$ and aborts.

We show that the probability that $\mathcal{H}_2$ outputs $\mathsf{binding\ abort}$ is negligible.

Claim 19. *Let ϵ be the probability that for some j, adversary $\mathcal{A}$ provides different openings to $M_{j,2}$. Then, ϵ is negligible in k.*

Proof We construct a non-uniform adversary $\mathcal{A}_{\mathsf{com}}$ that breaks the binding property of commitment scheme scom with probability ϵ. To show this, the adversary $\mathcal{A}_{\mathsf{com}}$ must succeed with probability ϵ in the following game: $\mathcal{A}_{\mathsf{com}}$ interacts with an external party, called the challenger, and acts as the sender in commitment scheme scom. If the commitment phase is successful, let $\hat{\alpha}$ be the transcript of the receiver in the commitment phase. The adversary succeeds if it can produce $(r, 0)$ and $(r', 1)$, both of which are valid decommitments with respect to $\hat{\alpha}$.

Let j be such that $\mathcal{A}$ queries $M_{j,2}$ twice on two distinct bits. The adversary $\mathcal{A}_{\mathsf{com}}$ works as follows: it sets up an execution between $\mathcal{Z}$, $\mathcal{A}$ and honest P_1 as in $\mathcal{H}_2$. Then it executes the j^{th} copy of scom with the challenger. If $\mathcal{A}$ queries $M_{j,2}$ only once, abort. Else, let $(r, 0)$ and $(r', 1)$ be the two queries. $\mathcal{A}_{\mathsf{com}}$ outputs $(r, 0)$ and $(r', 1)$.

By the hypothesis, with probability at least ϵ, $\mathcal{A}_{\mathsf{com}}$ outputs two different openings. As scom is computationally binding, ϵ must be negligible in k.

□

Hybrid $\mathcal{H}_3$: Same as above, except we modify the last message of S_2. We set $s'_b = s_b \bigoplus_{j=1}^{k}(z_j \oplus b_j \cdot r)$ and set $s'_{1\oplus b}$ to a random k-bit string.

We show that in $\mathcal{H}_2$, $s'_{1\oplus b}$ is uniformly distributed in $\{0,1\}^k$ given previous messages.

We first show that for every j, if b_j is $\mathcal{A}$'s input to $M_{j,1}$, then $s'_{1\oplus b_j}$ is uniformly distributed in $\{0,1\}^k$. Observe that $M_{j,1}$ partitions the domain $\{0,1\}^{5k}$ into 2^k partitions $S_1, \ldots, S_{2^k}$, such that for all $x, x' \in S_i$, $M_{j,1}(x) = M_{j,1}(x')$. For $x \in \{0,1\}^{5k}$, let $S(x)$ be the partition that x belongs to. Call a partition S_i "good" if $|S_i| \geq 2^{3k}$. We have the following claim.

Claim 20. *If x is uniformly chosen in $\{0,1\}^{5k}$, then the probability that it belongs to a good partition is at least $1 - 2^{-k}$.*

Proof As there are only 2^k partitions, we have, by the union bound,

$$\Pr[\, S(x) \text{ is bad }] < \frac{2^k \cdot 2^{3k}}{2^{5k}} = 2^{-k}.$$

Thus, the probability that x lies in a good partition is at least $1 - 2^{-k}$.

□

Thus, with probability negligibly close to 1, the random variable $\mathcal{U}_{5k}|M_{j,1}(\mathcal{U}_{5k})$ has min-entropy $3k$. Thus, by the leftover hash lemma, we have that,

$$\Delta((h_j, h_j(x_{1\oplus b_j})), (\mathcal{U}_{\mathcal{H}}, \mathcal{U}_k)) \leq 2^{-k-1}.$$

That is, $h_j(x_{1\oplus b_j})$ is distributed close to randomly in $\{0,1\}^k$. Thus, for each j, $\mathcal{A}$ learns $z_j \oplus b \cdot r$, while $z_j \oplus (1 \oplus b) \cdot r$ is completely random to it. Note that as $s'_0 = s_0 \oplus \bigoplus_{j=1}^{k} z_j$ and $s'_1 = s_1 \oplus \bigoplus_{j=1}^{k} z_j \oplus r$, $\mathcal{A}$ learns only s_b, while $s_{1\oplus b}$ is complete random to it.

Hybrid $\mathcal{H}_4$: This is the ideal world. Note that $\mathcal{H}_3$ is exactly the simulator S_2. The simulator extracts b as above, sends it to the ideal OT functionality, and obtains s_b. Then it sets $s'_b = s_b \bigoplus_{j=1}^{k}(z_j \oplus b_j \cdot r)$, and $s'_{1\oplus b}$ to a random value, and sends (s'_0, s'_1) to $\mathcal{A}$. This hybrid is identical to $\mathcal{H}_3$.

□

Thus, we have the following main theorem for this section.

Theorem 21. ***(Interactive UC-secure computation using stateless tokens.)*** *Let f be a (possibly reactive) polynomial-time computable functionality. Then, assuming one-way functions exist, there exists a computationally UC-secure interactive protocol which realizes f in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model. Furthermore, the protocol only makes a black-box use of the one-way function.*

CHAPTER 5

Physically Uncloneable Functions

5.1 Defining PUFs

In this section we follow definitions given in [BFSK11]. 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. The mathematical formalization of a PUF due to [BFSK11] is the following.

A PUF-family $\mathcal{P}$ is a pair of (not necessarily efficient) algorithms Sample and Eval, and is parameterized by the bound on the noise of PUF's response d_{noise} and the range of the PUF's output rg. Algorithm Sample abstracts the PUF fabrication process and works as follows. On input the security parameter, 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 of length rg. The output is guaranteed to have bounded noise d_{noise}, meaning that, when running $\mathsf{Eval}(1^n, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(n)$. Wlog, we assume that the challenge space of a PUF is a full set of strings of a certain length.

Definition 1 (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 sampleable 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 the paper 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 of PUFs. 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.

Definition 2 (Average min-entropy)**.** *The average min-entropy of the measurement* $\mathsf{PUF}(q)$ *conditioned on the measurements of challenges* $\mathcal{Q} = (q_1, \ldots, q_{poly(n)})$ *is defined by*

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

where the probability is taken over the choice of id *from* $\mathcal{I}_n$ *and the choice of possible PUF responses on challenge* q*. The term* $\mathsf{PUF}(\mathcal{Q})$ *denotes a sequence of random variables* $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_{poly(n)})$ *each corresponding to an evaluation of the PUF on challenge* q_k*, for* $1 \le k \le poly(n)$*.*

Definition 3 (Unpredictability). *A (rg, d_{noise})-PUF family $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ for security parameter n is $(d_{\mathsf{min}}(n), m(n))$-unpredictable if for any $q \in \{0,1\}^n$ and challenge list $\mathcal{Q} = (q_1, \ldots, q_{poly(n)})$, one has that, if for all $1 \leq k \leq poly(n)$ the Hamming distance satisfies $\mathsf{dis}_{\mathsf{ham}}(q, q_k) \geq d_{\mathsf{min}}(n)$, then the average min-entropy satisfies $\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) \geq m(n)$, where $\mathsf{PUF}(\mathcal{Q})$ denotes a sequence of random variables $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_{poly(n)})$ each corresponding to an evaluation of the PUF on challenge q_k. Such a PUF-family is called a $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$-PUF family.*

Fuzzy Extractors. The output of a PUF is noisy, that is, feeding it with the same challenge twice may yield distinct, but still close, responses. Fuzzy extractors of Dodis et al. [**?**] are applied to the outputs of the PUF to convert such noisy, high-entropy measurements into *reproducible* randomness.

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

Definition 4 (Fuzzy Extractors). *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 $(st, p) \overset{\$}{\leftarrow} \mathsf{FuzGen}(a)$ it holds that $st \leftarrow \mathsf{FuzRep}(a', p)$.

5.2 Modeling Malicious PUFs

We allow our adversaries to send malicious PUFs to honest parties[1]. 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

[1]Throughout this section, we assume the reader is familiar with the original UC PUF formulation of Brzuska et al. ([BFSK11], Section 4.2).

to be executed instead of an honest PUF. But, note that as the adversary is a PPT machine, the malicious code it specifies must also be PPT. However, PUFs are *not* modeled as PPT machines, so this places severe restrictions on the adversaries[2]. In particular, modeling malicious PUFs in this way would disallow the adversary from modifying honest PUFs (or more precisely, the honest PUF manufacturing process). Instead, we allow the adversary to specify a "malicious PUF family", that the ideal functionality uses. 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 Definition 1. 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. 5.1) 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.

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

[2]Observe that allowing the adversary to specify the malicious code enables the simulator to "rewind" the malicious PUF. However, in the model we use, such rewinding is not possible.

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. 5.1.

Allowing Adversary to Create PUFs. We deviate from the original formulation of $\mathcal{F}_{\mathsf{PUF}}$ of Brzuska et al. [BFSK11] 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. We thank Margarita Vald [Val12] for pointing out this issue. Also, it should be noted that the PUF set-up is non-programmable, but not *global* [CDPW07]. 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 [BFSK11] for honest PUFs, fails in the presence of malicious PUFs. Consider the OT protocol in Fig. 3 in [BFSK11]. 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 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. From this point on in this paper, we only talk of admissible families.

5.3 Other Definitions

5.3.1 Commitment Schemes

Definition 5 (Bit Commitment Scheme)**.** *A commitment scheme is a tuple of PPT algorithms* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *implementing the following two-phase functionality. Given to* C *an input* $b \in \{0, 1\}$*, in the first phase (*
emphcommitment phase) C *interacts with* R *to commit to the bit* b*, we denote this interaction as* $((c, d), c) \leftarrow \langle \mathsf{C}(\mathit{com}, b), \mathsf{R}(\mathit{recv}) \rangle$ *where* c *is the transcript of the commitment phase and* d *is the decommitment. In the second phase (*opening phase*)* C *sends* (b, d) *and* R *finally accepts or rejects according to* (c, b, d)*.*

$\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *is a commitment scheme if it satisfies the following properties.*

Completeness. *If* C *and* R *follow their prescribed strategy then* R *will always accept the commitment and the decommitment (with probability 1).*

Computational Hiding. *For every PPT* R^* *the ensembles* $\{\mathsf{view}_{\mathsf{R}^*}\ (\mathsf{C}(\mathit{com}, 0),\ \mathsf{R}^*)\ (1^n)\}_{n \in \mathbb{N}}$

and $\{\mathsf{view}_{\mathsf{R}^*}(\mathsf{C}(\mathit{com}, 0), \mathsf{R}^*)\ (1^n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable, where* $\mathsf{view}_{\mathsf{R}^*}$ $(\mathsf{C}(\mathit{com}, b),\ \mathsf{R}^*)$ *denotes the view of* R^* *in the commit stage interacting with* $\mathsf{C}(\mathit{com}, b)$*.*

Computational Binding. *For every PPT* C^**, there exists a negligible function* ϵ *such that the malicious sender* C^* *succeeds in the following game with probability at most* $\epsilon(n)$*: On security parameter* 1^n*,* C^* *interacts with R in the commit stage obtaining the transcript c . Then* C^* *outputs pairs* $(0, d_0)$ *and* $(1, d_1)$*, and succeeds if in the opening phase,* $\mathsf{R}(0, d_0, c) = \mathsf{R}(1, d_1, c) =$ *accept.*

When hiding (resp., binding) holds even against an unbounded adversary, then the commitment scheme enjoys statistical hiding (resp., binding).

It will be helpful to consider commitment schemes in which the committer and receiver take an additional common input, denoted by σ. This additional common input is drawn fresh for each execution from a specified distribution. In our case, this additional common input is always drawn from the uniform distribution of appropriate length. We will denote such commitment schemes with $\mathsf{Com} = (\mathsf{C}, \mathsf{R})(\sigma)$. The properties of Definition 5 are required to hold over the random choice of σ.

Definition 6 (Straight-line Equivocal Commitment Scheme.)**.** *A commitment scheme* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})(\sigma)$ *with common input* σ*, is a* straight-line equivocal commitment scheme *if there exists a straight-line strict polynomial-time simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *such that for any* $b \in \{0, 1\}$*, and for all PPT* R^**, the output of the following two experiments is computationally indistinguishable:*

Experiment $\boldsymbol{Exp}^{\mathsf{C}}_{\mathsf{R}^*}(n)$:	*Experiment* $\boldsymbol{Exp}^{\mathcal{S}}_{\mathsf{R}^*}(n)$:
$\sigma \xleftarrow{\$} \{0,1\}^{\ell(n)}$;	$(\tilde{\sigma}, \mathsf{state}_1) \xleftarrow{\$} \mathcal{S}_1(1^n)$;
$((c, d), c) \leftarrow \langle \mathsf{C}(\mathit{com}, b), \mathsf{R}^*(\mathit{recv})\rangle(\sigma)$;	$(\mathsf{state}_2, \tilde{c}) \leftarrow \langle \mathcal{S}_2(\mathsf{state}_1), \mathsf{R}^*(\mathit{recv})\rangle(\tilde{\sigma})$;
return $\mathsf{R}^*(\sigma, b, c, d)\rangle$;	$\tilde{d} \leftarrow \mathcal{S}_3(\sigma, \mathsf{state}_2, b)$; return $\mathsf{R}^*(\tilde{\sigma}, \tilde{c}, b, \tilde{d})$;

Note that in this definition, the verification of the receiver R is computed by using also the common input. Further, this definition can easily be extended to the setting where all parties have access to PUFs.

Definition 7 (Straight-line Extractable Commitment Scheme in the Malicious PUF model)**.** *A commitment scheme* $\mathsf{Com} = (\mathsf{C}, \mathsf{R})$ *is a* straight-line extractable commitment scheme in the malicious PUF model *if there exists a straight-line strict polynomial-time extractor* E *that, having on-line access to the queries made by any PPT malicious committer* C^* *to the PUFs sent by the honest receiver* R*, and running only the commitment phase, it outputs a bit* $b^\star$ *or the special symbol* $\perp$ *such that:*

- *(simulation) the views* $\mathsf{view}_{\mathsf{C}^*}(\mathsf{C}^*(\mathit{com}, \star), \mathsf{R}(\mathit{recv}))$ *and* $\mathsf{view}_{\mathsf{C}^*}(\mathsf{C}^*(\mathit{com}, \star), \mathsf{E})$ *are identical;*
- *(extraction) let* c *be the transcript obtained from the commitment phase run between* C^* *and* E*. If* E *outputs* $\perp$ *then the probability that* C^* *will provide an accepting decommitment is negligible.*
- *(binding) if* $b^\star \neq \perp$ *the probability that* C^* *decommit to a bit* $b \neq b^\star$ *is negligible.*

Remark 1. *The standard definition of extractable commitments in the plain model requires that, if the* commitment *is accepting then, probability that the extractor fails and outputs* $\perp$*, is negligible. Our definition is in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model, and straight-line extractability is achieved using access to* $\mathcal{F}_{\mathsf{PUF}}$*. Here, we require that, if the* decommitment *is accepting, then probability that the extractor fails and output* $\perp$ *is negligible. Therefore, to establish that an extractor fails, we have to consider the probability of success in the decommitment phase. To see why this definition is necessary, consider the following protocol. To commit to a bit* b*, the committer sends COM(b) to the receiver, then it queries a PUF (received from the receiver) with the opening of COM(b), and finally it commits to the response received from such PUF. Then in the decommitment phase, the committer has to open both commitments, and send the PUF back to the receiver, who accepts iff the openings are accepting and the response committed by the committer corresponds to the response of the PUF on input the opening of COM(b). In this case, a committer can always provide an accepting commitment, without querying the PUF (making the extractor output* $\perp$*). Indeed, it can just commit to junk. However, in the decommitment phase, the committer will not be able to open to the correct response, and the receiver will not accept. In such case, the extractor did not fail, since the committer did not actually commit to any value that could have been opened.*

5.3.2 Statistical Zero-Knowledge Argument of Knowledge

A polynomial-time relation R is a relation for which it is possible to verify in time polynomial in $|x|$ whether $R(x, w) = 1$. Let us consider an **NP**-language L and denote by R_L the corresponding polynomial-time relation such that $x \in L$ if and only if there exists w such that $R_L(x, w) = 1$. We will call such a w a *valid witness for* $x \in L$. We will denote by $\mathrm{Prob}_r[\ X\ ]$ the probability of an event X over coins r.

Interactive proof/argument systems with efficient prover strategies. An *interactive proof (resp., argument) system* for a language L is a pair of probabilistic polynomial-time interactive algorithms P and V, satisfying the requirements of *completeness* and *soundness*. Informally, completeness requires that for any $x \in L$, at the end of the interaction between P and V, where P has as input a valid witness for $x \in L$, V rejects with negligible probability. Soundness requires that for any $x \notin L$, for any (resp., any polynomial-sized) circuit P^*, at the end of the interaction between P^* and V, V accepts with negligible probability. We denote by $\mathsf{out}(\langle P(w), V\rangle(x))$ the output of the verifier V when interacting on common input x with prover P that also receives as additional input a witness w for $x \in L$. Moreover we denote by $\mathsf{out}(\langle P^*, V\rangle(x))$ the output of the verifier V when interacting on common input x with an adversarial prover P^*.

Formally, we have the following definition.

Definition 8. *A pair of interactive algorithms* $\langle P(\cdot), V(\cdot)\rangle(\cdot)$ *is an* interactive proof (resp., argument) system *for the language L, if V runs in probabilistic polynomial-time and*

1. *Completeness: For every $x \in L$, $|x| = n$, and for every **NP** witness w for $x \in L$*

$$\Pr[\,\mathsf{out}(\langle P(w), V\rangle(x) = 1\,] = 1.$$

2. *Soundness (resp. computational soundness): For every (resp., every polynomial-sized) circuit family $\{P_n^*\}_{n\in N}$ there exists a negligible function $\epsilon(\cdot)$ such that*

$$\Pr[\,\mathsf{out}(\langle P_n^*, V\rangle(x)) = 1\,] < \epsilon(|x|).$$

for every $x \notin L$ of size n.

Arguments of knowledge. Informally, an argument system is an argument of knowledge if for any probabilistic polynomial-time interactive algorithm P^* there exists a probabilistic algorithm called the extractor, such that 1) the expected running time of the extractor is polynomial-time regardless of the success probability of P^*; 2) if P^* has non-negligible probability of convincing a honest verifier for proving that $x \in L$, where L is an **NP** language, then the extractor with overwhelming probability outputs a valid witness for $x \in L$.

Zero knowledge. The classical notion of zero knowledge has been introduced in [GMR89]. 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 9. *An interactive argument system $\langle P(\cdot,\cdot), V(\cdot)\rangle$ for a language L is computational (resp., statistical, perfect)* zero-knowledge *if for all polynomial-time verifiers V^*, there exists an expected polynomial-time algorithm S such that the following ensembles are computationally (resp., statistically, perfectly) indistinguishable:*

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

$\mathcal{F}_{\mathsf{PUF}}$ 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 $\mathcal{S}$.

- – If this is the case, then turn into the 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, then turn into 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 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 $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ 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 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 $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(*))$ or $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \mathcal{S}, \mathsf{notrans})$.
  - If not, then turn into 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 into the 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 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 $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ has been stored. If not, return to the waiting state. Else, write this tuple to the input tape of P_i.

Figure 5.1: The ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ for malicious PUFs.

CHAPTER 6

UC from PUFs

In this chapter we present a construction for UC-secure commitment scheme in the malicious PUFs model, which yields UC-security for any PPT functionality via the [CLOS02] compiler.

We first recall some of the peculiarities of the PUFs model. A major difficulty when using PUFs, in contrast to say tamper-proof tokens, is that PUFs are *not programmable*. That is, the simulator can not simulate the answer of a PUF, and must honestly forward the queries to the $\mathcal{F}_{\mathsf{PUF}}$ functionality. The only power of the simulator is to observe 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, therefore allowing the simulator to extract such information in straight-line. In the malicious PUFs model the behaviour of a PUF created and 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).

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 [CO99] which builds upon Naor's scheme [Nao89]. Naor's scheme consists of two messages, where 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, to commit to bit b, the second message is $G(s) \oplus (r \wedge b^{|r|})$, where $G()$ is a PRG, and s a randomly chosen seed. The scheme has the property that if the string r is crafted appropriately, then the commitment is equivocal. [CO99] shows how this can be achieved by adding a coin-tossing phase before the commitment. The coin tossing of [CO99] 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. One extractable commitment from used by the receiver to commit to α must be statistically-hiding (this is necessary to prove binding). We denote such commitment as $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$. Another extractable commitmentt used by the sender to commit to its own bit 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 achieve extractability using the following technique.

Technique for Extracting the Bit Committed. The receiver creates a PUF and queries it with two randomly chosen challenges (q_0, q_1), obtaining the respective PUF-responses (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 PUF-response a_b. Note that the sender does not commit to the bit directly, but to the *answer* of the PUF. This ensures extractability. To decommit to b, the sender simply opens the commitment of the PUF-response sent before. Note that the receiver can check the authenticity of the PUF-response without having its own PUF back. The simulator can extract the bit by observing 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 technique ensures extractability. To additionally achieve statistically hiding and equivocality, we build protocol $\mathsf{Com}_{\mathsf{shext}}$ and $\mathsf{Com}_{\mathsf{equiv}}$ on top of this technique in different ways accordingly to the different properties that they achieve. The main difference is in the way the PUF-response a_b is committed.

In Protocol $\mathsf{Com}_{\mathsf{shext}}$, the sender $\mathsf{C}_{\mathsf{shext}}$ 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 given in Fig. 6.1. To commit to a N-bit string using $\mathsf{Com}_{\mathsf{shext}}$, it is sufficient to run the same protocol N times in parallel reusing the same PUF.

In Protocol $\mathsf{Com}_{\mathsf{equiv}}$ the PUF-response a_b is committed following Naor's commitment scheme. In $\mathsf{Com}_{\mathsf{equiv}}$, sender and receiver take as common input the vector $\bar{r} = r_1, \dots, r_l$ (l is size of a PUF-response a_b) which represent Naor's parameter decided in the coin-flipping phase. Earlier we said that the simulator can properly craft $\bar{r}$ so that it will be able to equivocate the commitment of a_b. However, due to the technique for extraction that we described above, being able to equivocate the commitment of a_b is not sufficient 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-response 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-response. 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-response and equivocate the commitment so to open to such PUF-response. 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-response, thus the size of Naor's parameter $\bar{r}$ is $N = (3n)l$. A graphic representation of Protocol $\mathsf{Com}_{\mathsf{equiv}}$ is given in Fig. 6.2.

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 formally described in Fig. 6.4.

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.* We finally stress that we do not assume authenticated PUF delivery. Namely, the privacy of the honest party is preserved even if the adversary interferes with the delivery process of the honest PUFs (e.g., by replacing the honest PUF).

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. 6.4, is a UC-secure commitment scheme in the*

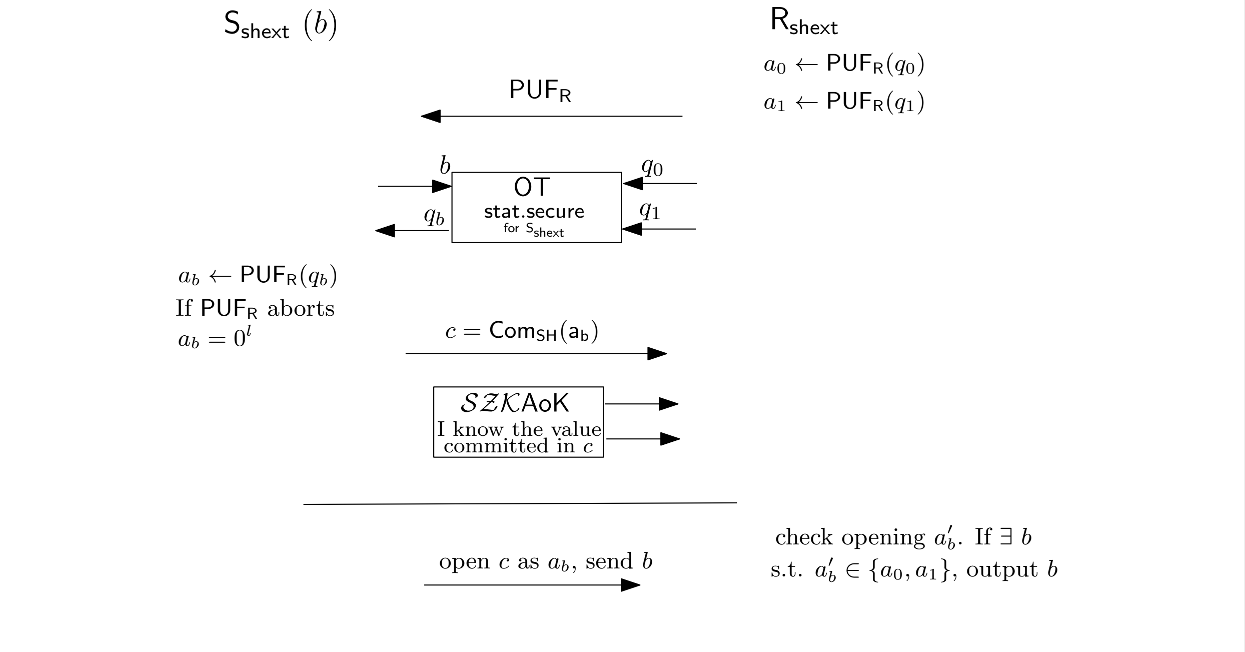

$\mathsf{S}_{\mathsf{shext}}\ (b)$
$\mathsf{R}_{\mathsf{shext}}$
$a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$
$a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$
$\mathsf{PUF}_{\mathsf{R}}$
b
q_0
OT
stat.secure
for $\mathsf{S}_{\mathsf{shext}}$
q_b
q_1
$a_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_b)$
If $\mathsf{PUF}_{\mathsf{R}}$ aborts
$a_b = 0^l$
$c = \mathsf{Com}_{\mathsf{SH}}(\mathsf{a_b})$
$\mathcal{SZK}$AoK
I know the value committed in c
check opening a'_b. If $\exists\ b$
s.t. $a'_b \in \{a_0, a_1\}$, output b
open c as a_b, send b

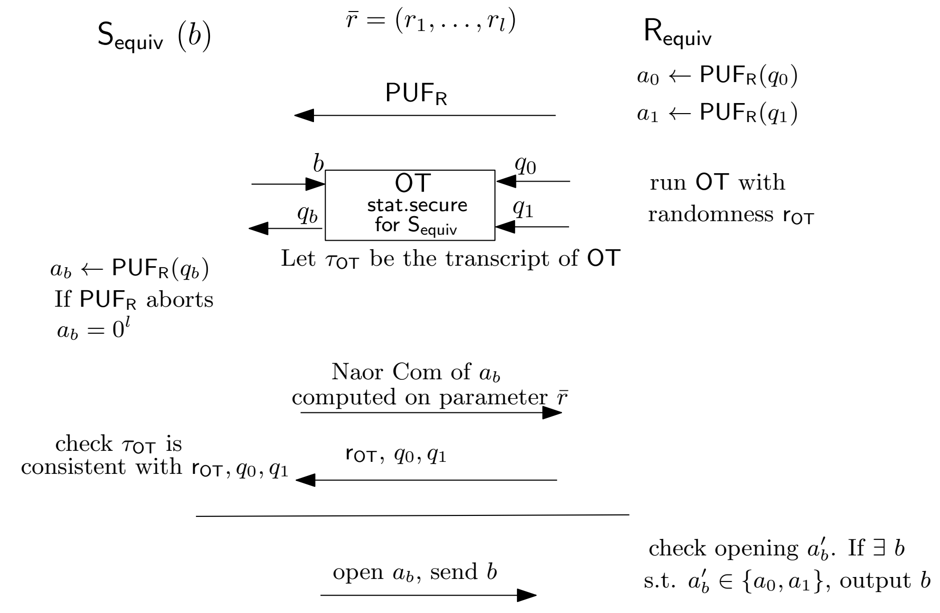
Figure 6.1: Straight-line Extractable Statistically-hiding Bit Commitment $\mathsf{C}_{\mathsf{shext}}$

malicious PUFs model.

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 we can not reuse the *same* PUF when multiple commitments are executed *concurrently*[1]. The reason is that, in both sub-protocols $\mathsf{Com}_{\mathsf{shext}}, \mathsf{Com}_{\mathsf{equiv}}$, in the opening phase the sender 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 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,

[1]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.



$\mathsf{S}_{\mathsf{equiv}}\ (b)$
$\bar{r} = (r_1, \dots, r_l)$
$\mathsf{R}_{\mathsf{equiv}}$
$a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$
$a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$
$\mathsf{PUF}_{\mathsf{R}}$
b
q_0
OT
stat.secure
for $\mathsf{S}_{\mathsf{equiv}}$
q_b
q_1
run OT with
randomness r_{OT}
Let τ_{OT} be the transcript of OT
$a_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_b)$
If $\mathsf{PUF}_{\mathsf{R}}$ aborts
$a_b = 0^l$
Naor Com of a_b
computed on parameter $\bar{r}$
check τ_{OT} is
consistent with $\mathsf{r}_{\mathsf{OT}}, q_0, q_1$
r_{OT}, q_0, q_1
check opening a'_b. If $\exists\ b$
s.t. $a'_b \in \{a_0, a_1\}$, output b
open a_b, send b

Figure 6.2: Straight-line Extractable Statistically-binding Equivocal Bit Commitment $\mathsf{Com}_{\mathsf{equiv}}$

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 [CLOS02], which only needs a UC secure implementation of the $\mathcal{F}_{\mathsf{mcom}}$ functionality.

6.1 Component Commitment Schemes

6.1.1 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, $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ 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 ($\mathcal{SZK}\mathsf{AoK}$) for the following relation: $\mathcal{R}_{\mathsf{com}} = \{(c, (s, d))$ such that $\mathsf{R}_{\mathsf{SH}}(c, s, d) = 1\}$. A pictorial description of protocol $\mathsf{Com}_{\mathsf{shext}} = (\mathsf{C}_{\mathsf{shext}}, \mathsf{R}_{\mathsf{shext}})$ was given in Fig. 6.1. In Fig. 6.5 we provide the formal specification.

Theorem 2. *If* $\mathsf{Com}_{\mathsf{SH}} = (\mathsf{C}_{\mathsf{SH}}, \mathsf{R}_{\mathsf{SH}})$ *is a statistically-hiding commitment scheme,* $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$

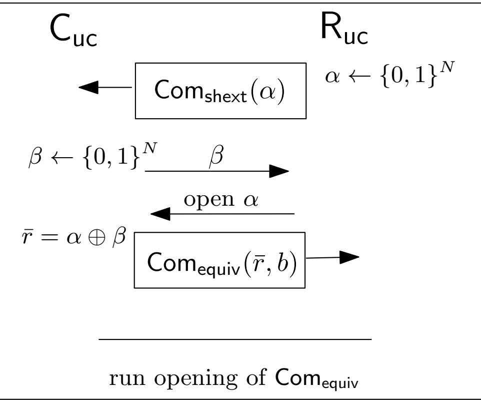

C_{uc}
R_{uc}
$\mathsf{Com}_{\mathsf{shext}}(\alpha)$
$\alpha \leftarrow \{0,1\}^N$
$\beta \leftarrow \{0,1\}^N$
β
open α
$\bar{r} = \alpha \oplus \beta$
$\mathsf{Com}_{\mathsf{equiv}}(\bar{r}, b)$
run opening of $\mathsf{Com}_{\mathsf{equiv}}$

Figure 6.3: 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 $\alpha \overset{\$}{\leftarrow} \{0,1\}^N$; commit to α by running $((c_\alpha, d_\alpha), c_\alpha) \leftarrow \langle \mathsf{C}_{\mathsf{shext}}(\mathtt{com}, \alpha), \mathsf{R}_{\mathsf{shext}}(\mathtt{recv})\rangle$ with C_{uc}.
2. C_{uc} sends $\beta \overset{\$}{\leftarrow} \{0,1\}^N$ 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 $((c_{\mathsf{bit}}, d_{\mathsf{bit}}), c_{\mathsf{bit}}) \leftarrow \langle \mathsf{C}_{\mathsf{equiv}}(\mathtt{com}, b), \mathsf{R}_{\mathsf{equiv}}(\mathtt{recv})\rangle(\alpha \oplus \beta)$ with R_{uc}.

Decommitment Phase

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

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

Figure 6.4: Computational UC Commitment Scheme $(\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$.

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

Commitment Phase

$\mathsf{R_{shext}}$: Initialize $\mathsf{PUF_R}$.

1. obtain $a_0 \leftarrow \mathsf{PUF_R}(q_0)$, $a_1 \leftarrow \mathsf{PUF_R}(q_1)$, 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 $\mathsf{PUF_R}$ to $\mathsf{C_{shext}}$.

$\mathsf{R_{shext}} \Leftrightarrow \mathsf{C_{shext}}$: (Statistical OT phase)

$\langle \mathsf{S_{OT}}(q_0, q_1), \mathsf{R_{OT}}(b)\rangle$ is run by $\mathsf{R_{shext}}$ as $\mathsf{S_{OT}}$ with input (q_0, q_1), and $\mathsf{C_{shext}}$ as $\mathsf{R_{OT}}$ with input b. Let q'_b be the local output of $\mathsf{C_{shext}}$.

$\mathsf{C_{shext}}$: $a'_b \leftarrow \mathsf{PUF_R}(q'_b)$. If $\mathsf{PUF_R}$ aborts, $a'_b \xleftarrow{\$} \{0,1\}^l$.

$\mathsf{C_{shext}} \Leftrightarrow \mathsf{R_{shext}}$: (Statistically Hiding Commitment)

$((c, d), c) \leftarrow \langle \mathsf{C_{SH}}(\mathtt{com}, a'_b), \mathsf{R_{SH}}(\mathtt{recv})\rangle$ is run by $\mathsf{C_{shext}}$ as $\mathsf{C_{SH}}$ to commit to a'_b, and by $\mathsf{R_{shext}}$ as $\mathsf{R_{SH}}$.

$\mathsf{C_{shext}} \Leftrightarrow \mathsf{R_{shext}}$: ($\mathcal{SZK}\mathsf{AoK}$)

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

Decommitment Phase

$\mathsf{C_{shext}}$: if $\mathsf{PUF_R}$ did not abort, send opening (d, a'_b, b) to $\mathsf{R_{shext}}$.

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

Figure 6.5: Statistically Hiding Straight-Line Extractable Bit Commitment Scheme $(\mathsf{C_{shext}}, \mathsf{R_{shext}})$.

is a statistical receiver OT protocol and (P, V) is a $\mathcal{SZK}\mathsf{AoK}$, then $\mathsf{Com_{shext}}$ is a statistically hiding straight-line extractable bit commitment scheme in the malicious PUFs model.

Proof

Completeness. Before delivering its own PUF $\mathsf{PUF_R}$, $\mathsf{R_{shext}}$ 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, $\mathsf{C_{shext}}$ has to commit to the output a_b of $\mathsf{PUF_R}$.

By the completeness of the OT protocol, $\mathsf{C_{shext}}$ obtains the query q_b corresponding to its secret bit. Then $\mathsf{C_{shext}}$ queries $\mathsf{PUF_R}$ with q_b and commits to the response a'_b running $\mathsf{C_{SH}}$. 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 $\mathsf{R^*_{shext}}$ it holds that:

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

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 $\mathsf{R^*_{shext}}$ in the commitment phase consists of the transcript of the execution of the OT protocol $(\mathsf{S_{OT}}, \mathsf{R_{OT}})$, 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 $\mathsf{PUF^*_R}$ 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, $\mathsf{C}_{\mathsf{shext}}$ proceeds as in H_1, excepts that it runs C_{SH} to commit to a random string s instead of a'_0. By the statistically hiding property of protocol $\mathsf{Com}_{\mathsf{SH}}$, hybrids H_1 and H_2 are statistically indistinguishable.

H_3: In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ 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 $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$, hybrids H_2 and H_3 are statistically indistinguishable.

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

H_6: In this hybrid, $\mathsf{C}_{\mathsf{shext}}$ 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 7. 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,

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

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

Lemma 22 ($\Pr[\, \textbf{Fail1} \,]$ is negligible)**.** *If* $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ *is an Oblivious Transfer protocol, then* $\Pr[\, \textbf{\textit{Fail1}} \,]$ *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 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 $\mathsf{PUF}_{\mathsf{R}}$ to $\mathsf{C}^*_{\mathsf{shext}}$, then it runs the OT protocol forwarding the messages received from the external sender 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 $\mathsf{PUF}_{\mathsf{R}}$. 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 $(\mathsf{S}_{\mathsf{OT}}, \mathsf{R}_{\mathsf{OT}})$ is a stand-alone secure OT protocol, δ must be negligible.

□

Lemma 23 (Pr [**Fail2**] is negligible)**.** *Assume that τ_c is an accepting transcript. If* $\mathsf{Com}_{\mathsf{SH}} = (\mathsf{C}_{\mathsf{SH}}, \mathsf{R}_{\mathsf{SH}})$ *is a commitment scheme and if* (P, V) *is a* $\mathcal{SZK}\mathsf{AoK}$ *then* Pr [***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 $\mathsf{R}_{\mathsf{shext}}$ accepts, i.e., the opening (d) of the statistically hiding commitment is valid *and* corresponds to an answer (a'_b) of $\mathsf{PUF}_{\mathsf{R}}$ upon one of the queries played by the $\mathsf{R}_{\mathsf{shext}}$ in the OT protocol. Formally, $\mathsf{R}_{\mathsf{SH}}(c, a'_b, d) = 1$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b$.

Toward a contradiction, assume that Pr [**Fail2**] $= \delta$ and is not-negligible. Recall that the event **Fail2** happens when $\mathsf{C}^*_{\mathsf{shext}}$ successfully completed the commitment phase, without querying $\mathsf{PUF}_{\mathsf{R}}$ 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 $\mathsf{PUF}_{\mathsf{R}}$. By the unpredictability of $\mathsf{PUF}_{\mathsf{R}}$ we have that this case has negligible probability to happen.
2. $\mathsf{C}^*_{\mathsf{shext}}$ queries $\mathsf{PUF}_{\mathsf{R}}$ in the *decommitment* phase to obtain the value a'_b to be opened. Thus $\mathsf{C}^*_{\mathsf{shext}}$ opens commitment 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 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 $(c, \cdot) \in \mathcal{R}_{\mathtt{com}}$, $\mathsf{C}^*_{\mathsf{SH}}$ 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 $c \in \mathcal{R}_{\mathtt{com}}$ w.h.p. If the witness extracted is not a valid decommitment of 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 23, 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 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 PUF_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 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 PUF_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 PUF_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 PUF_R with query q_{b_0} only; 2) in decommitment phase $\mathsf{C}^*_{\mathsf{shext}}$ queried PUF_R with q_{b_1}, let a_{b_1} be the answer; 3)$\mathsf{C}^*_{\mathsf{shext}}$ opens the commitment c (that is the commitment of the answer of PUF_R received in the commitment phase), as a_{b_1}, but 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 24. *Protocol* $\mathsf{Com}_{\mathsf{shext}}$ *is close under parallel repetition using the same PUF.*

Proof [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 6.3 we use the same notation $\mathsf{Com}_{\mathsf{shext}}$ to refer to a string commitment scheme.

6.1.2 Statistically Binding Straight-line Extractable and Equivocal Commitment Scheme

Let $l = rg(n)$ be the range of the PUF, $(\mathsf{S_{OT}}, \mathsf{R_{OT}})$ be a statistical receiver OT protocol and let $G : \{0,1\}^n \rightarrow \{0,1\}^{3n}$ 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 $\mathsf{Com_{equiv}} = (\mathsf{C_{equiv}}, \mathsf{R_{equiv}})$ is depicted in Fig. 6.6. A graphical representation was provided in Fig. 6.2.

Theorem 3. *If G is a PRG and $(\mathsf{S_{OT}}, \mathsf{R_{OT}})$ is statistical receiver OT protocol, then $\mathsf{Com_{equiv}} = (\mathsf{C_{equiv}}, \mathsf{R_{equiv}})$ 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 fuzzy extractors. To commit to the bit b, the sender $\mathsf{C_{equiv}}$ is required to commit to the answer of $\mathsf{PUF_R}$ on input q_b. Therefore, $\mathsf{C_{equiv}}$ 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 $\mathsf{C_{equiv}}$ goes through. In the decommitment phase, the response consistency property along with correctness of Naor, allow the receiver $\mathsf{R_{equiv}}$ to obtain the string a_b and in therefore the bit decommitted to by $\mathsf{C_{equiv}}$.

Straight-line Extractability.

<u>Extractor E</u>

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

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 $(q_0, q_1) \overset{\$}{\leftarrow} \{0,1\}^n$.
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 $\mathsf{ran}_{\mathsf{OT}} \overset{\$}{\leftarrow} \{0,1\}^*$.

$\mathsf{R}_{\mathsf{equiv}} \Leftrightarrow \mathsf{C}_{\mathsf{equiv}}$: (OT phase)

$\langle \mathsf{S}_{\mathsf{OT}}(q_0, q_1), \mathsf{R}_{\mathsf{OT}}(b) \rangle$ 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 R_{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, $a'_b \overset{\$}{\leftarrow} \{0,1\}^l$.
2. for $1 \leq i \leq l$, pick $s_i \overset{\$}{\leftarrow} \{0,1\}^n$, $c_i = G(s_i) \oplus (r_i \wedge a'_b[i])$.
3. send $c_1, \ldots, c_l$ to $\mathsf{R}_{\mathsf{equiv}}$.

$\mathsf{R}_{\mathsf{equiv}}$: upon receiving $c_1, \ldots, c_l$, 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 $(c_i = G(s_i) \oplus (r_i \wedge a'_b[i])$ and $\mathsf{FuzRep}(a'_b, p_b) = st_b)$ then accept. Else reject.

Figure 6.6: Statistically Binding Straight-line Extractable and Equivocal Commitment $(\mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}})$.

abort. Else, let $\mathcal{Q}_{\mathsf{precom}}$ be the set of queries asked by $\mathsf{C}^*_{\mathsf{equiv}}$ to $\mathsf{PUF}_{\mathsf{R}}$ *before* sending the commitments $c_1, \ldots, c_l$ 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 $\mathsf{dis}_{\mathsf{ham}}(q_0, q'_0) \leq d_{\mathsf{min}}$ and $\mathsf{dis}_{\mathsf{ham}}(q_1, q'_1) \leq d_{\mathsf{min}}$, output $b^\star = \perp$.

Fail2. If for all $q' \in \mathcal{Q}_{\mathsf{precom}}$ 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 $q' \in \mathcal{Q}_{\mathsf{precom}}$ such that $\mathsf{dis}_{\mathsf{ham}}(q_0, q') \leq d_{\mathsf{min}}$ then output $b^\star = 0$;

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

Finally sends $\mathsf{ran}_{\mathsf{OT}}, q_0, q_1$ 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,

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

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 22. Given that the common parameter $\bar{r}$ is uniformly chosen, we have that the Naor's commitments (i.e., $c_1, \ldots, c_l$) 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 $\mathsf{R}^*_{\mathsf{equiv}}$.

$\underline{\mathcal{S}_1}$. $(\bar{r} = r_1, \ldots, r_l, \mathsf{state}_1) \leftarrow \mathcal{S}_1(1^{ln})$:

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 $r_1, \ldots, r_l$, $\mathsf{state}_1 = \{s_0^i, s_1^i\}_{i \in l}$;

$\underline{\mathcal{S}_2}$. $(\mathsf{state}_2) \leftarrow \mathcal{S}_2(\mathsf{state}_1)$:

- 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 $i = 1, \ldots, l$, $\tilde{c}_i \leftarrow G(s_0^i)$. Send $\tilde{c}_1, \ldots, \tilde{c}_l$ 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 $\mathsf{state}_2 = \{\mathsf{state}_1, (q'_0, q'_1)\}$.

$\underline{\mathcal{S}_3}$. $\mathcal{S}_3(\mathsf{state}_2, b)$:

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

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 $c_1, \ldots, c_l$ are computed as in $\mathcal{S}_2$, that is, for all i, c_i corresponds to an evaluation of the PRG i.e., $c_i = G(s_0^i)$, 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 $\mathbf{Exp}^{\mathcal{S}}_{\mathsf{R}^*_{\mathsf{equiv}}}$.

$\square$

$\square$

6.2 Proof of Security of UC Commitment

In this section we show that protocol $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{C}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ is UC-secure, by showing a PPT ideal world adversary Sim such that for all PPT environment $\mathcal{Z}$, 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 $\mathsf{E}_{\mathsf{C}_{\mathsf{shext}}}$ associated to the protocol $\mathsf{Com}_{\mathsf{shext}}$.

Simulator 1.

Commitment Phase

- Run $(\bar{r}, \mathsf{state}_1) \leftarrow \mathcal{S}_1(1^{ln})$.
- Execute protocol $\mathsf{Com}_{\mathsf{shext}}$ by running the associated extractor $\mathsf{E}_{\mathsf{C}_{\mathsf{shext}}}$. If the output of the extractor is $\bot$, then abort. Else, let $\alpha^\star$ be the string extracted by $\mathsf{E}_{\mathsf{C}_{\mathsf{shext}}}$.

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 $\mathcal{S}_2(\mathsf{state}_1)$, and obtain state_2 as local output.

Decommitment Phase

- On input the bit b. Execute the decommitment phase of protocol $\mathsf{Com}_{\mathsf{equiv}}$ by running $\mathcal{S}_3(\mathsf{state}_2, b)$.
- Output whatever R_{uc} outputs.

Lemma 26. *For all PPT real-world malicious receiver $\mathsf{R}^*_{\mathsf{uc}}$, for all PPT adversary $\mathcal{Z}$, it holds that:*

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

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, $\mathcal{Z}$ 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 $\mathcal{Z}$.

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 $\mathsf{E}_{\mathsf{equiv}}$ 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 $\mathsf{E}_{\mathsf{equiv}}$. If the extrac-

tor aborts, abort. Else, let $b^\star$ the output of $\mathsf{E}_{\mathsf{equiv}}$. Send $(\mathsf{commit}, \mathsf{sid}, \mathsf{C}_{\mathsf{equiv}}, \mathsf{R}_{\mathsf{equiv}}, b^\star)$ to $\mathcal{F}_{\mathsf{com}}$

Lemma 27. *For all PPT real-world malicious committer $\mathsf{C}^*_{\mathsf{uc}}$, for all PPT adversary $\mathcal{Z}$, it holds that:*

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

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 $\mathsf{E}_{\mathsf{equiv}}$, 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.

$\square$

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[BCS96] Gilles Brassard, Claude Crépeau, and Miklos Santha. Oblivious transfers and intersecting codes. *IEEE Transactions on Information Theory*, 42(6):1769–1780, 1996.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, pages 479–488, 1996.

[BFSK11] 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.

[BG89] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority (extended announcement). In *FOCS*, pages 468–473. IEEE, 1989.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[Can01a] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[Can01b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CDPW07] 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.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, Lecture Notes in Computer Science, pages 19–40. Springer, 2001.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. pages 545–562, 2008.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. pages 494–503, 2002.

[CO99] Giovanni Di Crescenzo and Rafail Ostrovsky. On concurrent zero-knowledge with pre-processing. In *CRYPTO*, pages 485–502, 1999.

[CW79] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[GHY87] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In *CRYPTO*, pages 135–155, 1987.

[GKR08] Shafi Goldwasser, Yael T. Kalai, and Guy. N. Rothblum. One-time programs. In *Advances in Cryptology – CRYPTO'08*, volume 5157, pages 39–56, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SICOMP*, 18(6):186–208, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, Cambridge, UK, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications.* Cambridge University Press, Cambridge, UK, 2004.

[GV87] Oded Goldreich and Ronen Vainish. How to solve any protocol problem - an efficiency improvement. In *CRYPTO*, pages 73–86, 1987.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HR07] Iftach Haitner and Omer Reingold. Statistically-hiding commitment from any one-way function. In *STOC*, pages 1–10, 2007.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. pages 572–591, 2008.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, Lecture Notes in Computer Science, pages 115–128. Springer, 2007.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. pages 20–31, 1988.

[Kil90] Joe Kilian. *Uses of Randomness in Algorithms and Protocols.* MIT Press, 1990.

[Kus92] Eyal Kushilevitz. Privacy and communication complexity. *SIAM J. Discrete Math.*, 5(2):273–284, 1992.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[MS08] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. pages 527–544, 2008.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness. In *CRYPTO*, pages 128–136, 1989.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer, 1981.

[Val12] Margarita Vald. Private Communication, 2012.

[Yao86] A. Yao. How to generate and share secrets. In *FOCS*, pages 162–167, 1986.