Unconditionally Secure and Universally Composable Commitments from Physical Assumptions

Ivan Damgård*
Aarhus University, Denmark

Alessandra Scafuro†
UCLA, USA

Abstract

We present a constant-round unconditional black-box compiler that transforms any ideal (i.e., statistically-hiding and statistically-binding) straight-line extractable commitment scheme, into an extractable and equivocal commitment scheme, therefore yielding to UC-security [9]. We exemplify the usefulness of our compiler by providing two (constant-round) instantiations of ideal straight-line extractable commitment based on (malicious) PUFs [37] and *stateless* tamper-proof hardware tokens [27], therefore achieving the first unconditionally UC-secure commitment with malicious PUFs and stateless tokens, respectively. Our constructions are secure for adversaries creating arbitrarily malicious stateful PUFs/tokens.

Previous results with malicious PUFs used either computational assumptions to achieve UC-secure commitments or were unconditionally secure but only in the indistinguishability sense [37]. Similarly, with stateless tokens, UC-secure commitments are known only under computational assumptions [13, 25, 15], while the (not UC) unconditional commitment scheme of [24] is secure only in a weaker model in which the adversary is not allowed to create stateful tokens.

Besides allowing us to prove feasibility of unconditional UC-security with (malicious) PUFs and stateless tokens, our compiler can be instantiated with any ideal straight-line extractable commitment scheme, thus allowing the use of various setup assumptions which may better fit the application or the technology available.

Keywords: UC-security, hardware assumptions, unconditional security, commitment scheme.

1 Introduction

Unconditional security guarantees that a protocol is secure even when the adversary is unbounded. While it is known how to achieve unconditional security for multi-party functionalities in the plain model assuming honest majority [4, 14], obtaining unconditionally secure *two-party* computation is impossible in the plain model. In fact, for all non-trivial two-party functionalities, achieving unconditional security requires some sort of (physical) setup assumption.

Universally composable (UC) security [9] guarantees that a protocol is secure even when executed concurrently with many other instances of any arbitrary protocol. This strong notion captures the

*The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed.

†Work done while visiting Aarhus University. Research supported in part by NSF grants CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS- 1136174; and Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392.

real world scenarios, where typically many applications are run concurrently over the internet, and is therefore very desirable to achieve. Unfortunately, achieving UC-security in the plain model is impossible [11].

Hence, constructing 2-party protocols which are unconditionally secure or universally composable requires the employment of some setup. One natural research direction is to explore which setup assumptions suffice to achieve (unconditional) UC-security, as well as to determine whether (or to what extent) we can *reduce* the amount of trust in a third party. Towards this goal, several setup assumptions have been explored by the community.

In [12] Canetti et. al show that, under computational assumptions, any functionality can be UC-realized assuming the existence of a trusted Common Reference String (CRS). Here, the security crucially relies on the CRS being honestly sampled. Hence, security in practice would typically rely on a third party sampling the CRS honestly and security breaks down if the third party is not honest. Similar arguments apply to various assumptions like "public-key registration" services [3, 10].

Another line of research explores "physical" setup assumptions. Based on various types of noisy channels, unconditionally secure Bit Commitment (BC) and Oblivious Transfer (OT) can be achieved [16, 18] for two parties, but UC security has not been shown for these protocols and in fact seems non-trivial to get for the case of [18].

In [27] Katz introduces the assumption of the existence of tamper-proof hardware tokens. The assumption is supported by the possibility of implementing tamper-proof hardware using current available technology (e.g., smart cards). A token is defined as a physical device (a wrapper), on which a player can upload the code of any functionality, and the assumption is that any adversary cannot tamper with the token. Namely, the adversary has only black-box access to the token, i.e., it cannot do more then observing the input/output characteristic of the token. The main motivation behind this new setup assumption is that it allows for a *reduction of trust*. Indeed in Katz's model tokens are not assumed to be trusted (i.e., produced by a trusted party) and the adversary is allowed to create a token that implements an arbitrary malicious function instead of the function dictated by the protocol. (However, it is assumed that once the token is sent away to the honest party, it cannot communicate with its creator. This assumption is necessary, as otherwise we are back to the plain model). A consequence of this model is that the security of a player now depends only on its *own token* being good and holds even if tokens used by other players are not genuine! This new setup assumptions has gained a lot of interest and several works after [27] have shown that unconditional UC-security is possible [32, 25], even using a single *stateful* token [22, 23]. Note that a stateful token, in contrast with a *stateless* token, requires an updatable memory that can be subject to reset attacks. Thus, ensuring tamper-proofness for a stateful token seems to be more demanding than for a stateless token, and hence having protocols working with stateless tokens is preferable.

However, the only constructions known for stateless tokens require computational assumptions [13, 30, 25, 15] and a non-constant number of rounds (if based on one-way functions only). In fact, intuitively it seems challenging to achieve unconditional security with stateless tokens: A stateless token runs always on the same state, thus an unbounded adversary might be able to extract the secret state after having observed only a polynomial number of the token's outputs. This intuition is confirmed by [24] where it is proved that unconditional OT is impossible using stateless tokens. On the positive side, [24] shows an unconditional commitment scheme (not UC) based on stateless tokens, but the security of the scheme holds only if the adversary is not allowed to create malicious stateful tokens. This is in contrast with the standard tamper-proof hardware model, where the adversary is allowed to construct any arbitrary malicious (hence possibly stateful) token. Indeed, it seems difficult in practice to detect whether an adversary sends a stateless or a stateful token. Therefore, the question of achieving unconditional commitments (UC-secure or not) in the standard stateless token model (where an adversary possibly plays with stateful tokens) is still open.

In this work we provide a positive answer showing the first UC-secure unconditional commitment scheme with stateless tokens.

Following the approach of [27], Brzuska et al. in [7] propose a new setup assumption for achieving UC security, which is the existence of Physically Uncloneable Functions (PUFs). PUFs have been introduced by Pappu in [39, 38], and since then have gained a lot of interest for cryptographic applications [2, 43, 1, 41]. A PUF is a physical noisy source of randomness. In other words a PUF is a device implementing a function whose behavior is unpredictable even to the manufacturer. The reason is that even knowing the exact manufacturing process there are parameters that cannot be controlled, therefore it is assumed infeasible to construct two PUFs with the same challenge-response behavior. A PUF is noisy in the sense that, when queried twice with the same challenge, it can output two different, although close, responses. Fuzzy extractors are applied to PUF's responses in order to reproduce a unique response for the same challenge. The "PUF assumption" consists in assuming that PUFs satisfy two properties: 1) unpredictability: the distribution implemented by a PUF is unpredictable. That is, even after a polynomial number of challenge/response pairs have been observed, the response on any new challenge (sufficiently far from the ones observed so far) is unpredictable; this property is *unconditional*; 2) uncloneability: as a PUF is the output of a physical uncontrollable manufacturing process, it is assumed that creating two identical PUFs is hard even for the manufacturer. This property is called hardware uncloneability. Software uncloneability corresponds to the hardness of modeling the function implemented by the PUF and is enforced by unpredictability (given that the challenge/response space of the PUF is adequately large). Determining whether (or to what extent) current PUF candidates actually satisfy the PUF assumption is an active area of research (e.g., [28, 5]) but is out of the scope of this work. For a survey on PUF's candidates the reader can refer to [31], while a security analysis of silicon PUFs is provided in [28].

Designing PUF-based protocols is fundamentally different than for other hardware tokens. This is due to the fact that the functional behavior of a PUF is unpredictable even for its creator. Brzuska et al. modeled PUFs in the UC-setting by formalizing the ideal PUF functionality. They then provided constructions for Unconditional UC Oblivious Transfer and Bit Commitment. However, their UC-definition of PUFs assumes that all PUFs are *trusted*. Namely, they assume that even a malicious player creates PUFs honestly, following the prescribed generation procedure. This assumption seems too optimistic as it implies that an adversary must not be capable of constructing hardware that "looks like" a PUF but that instead computes some arbitrary function. The consequence of assuming that all PUFs are trusted is that the security of a player depends on the PUFs created by other players. (Indeed, in the OT protocol of [7], if the receiver replaces the PUF with hardware implementing some predictable function, the security of the sender is violated).

In [37] Ostrovsky et al. extend the ideal PUF functionality of [7] in order to model the adversarial behavior of creating and using "malicious PUFs". A malicious PUF is a physical device for which the security properties of a PUF are not guaranteed. As such, it can be a device implementing *any* function chosen by the adversary, so that the adversary might have full control on the answers computed by its own "PUF". Similarly to the hardware-token model, a malicious PUF cannot communicate with the creator once is sent away. A malicious PUF can, of course, be stateful. The major advantage of the malicious PUF model is that the security of a player depends only on the goodness of its own PUFs. Obviously, the price to pay is that protocols secure in presence of malicious PUFs are more complex than protocols designed to deal only with honest PUFs. Nevertheless, [37] shows that even with malicious PUFs it is possible to achieve UC-secure computations relying on computational assumptions. They also show an unconditional commitment scheme which is secure only in the indistinguishability sense. Achieving unconditional UC-secure commitments (and general secure computations) is left as an open problem in [37].

In this paper, we give a (partial) positive answer to this open problem by providing the first construction of unconditional UC-secure Bit Commitment in the malicious PUFs model. Whether unconditional OT (and thus general secure computation) is possible with malicious PUFs is still an interesting open question. Intuitively, since PUFs are stateless devices, one would think to apply the arguments used for the impossibility of unconditional OT with stateless tokens [24]. However, due to the unpredictability property of PUFs which holds unconditionally, such arguments do not carry through. Indeed, as long as there is at least one honest PUF in the system, there is enough entropy, and this seems to defeat the arguments used in [24]. On the other hand, since a PUF is in spirit just a “random function”, it is not clear how to implement the OT functionality when only one of the players uses honest PUFs.

Van Dijk and Rührmair in [20] also consider adversaries who create malicious PUFs, that they call “bad PUFs” and they consider only the stand-alone setting. They show that unconditional OT is impossible in the bad PUF model but this impossibility proof works assuming that also honest parties play with bad PUFs. Thus, such impossibility proof has no connection to the question of achieving unconditional OT in the malicious PUF model (where honest parties play with honest PUFs).

Our Contribution. In this work we provide a tool for constructing UC-secure commitments given any straight-line extractable commitment. This tool allows us to show feasibility results for unconditional UC-secure protocols in the stateless token model and in the malicious PUF model. More precisely, we provide an unconditional black-box compiler that transforms any ideal (i.e., statistically hiding and binding) straight-line extractable commitment into a UC-secure commitment. The key advantage of such compiler is that one can implement the ideal extractable commitment with the setup assumption that is more suitable to the application and the technology available.

We then provide an implementation of the ideal extractable commitment scheme in the malicious PUFs model of [37]. Our construction builds upon the (stand-alone) unconditional BC scheme shown in [37] [1] which is not extractable. By plugging our extractable commitment scheme in our compiler we obtain the first unconditional UC-secure commitment with malicious PUFs.

We then construct ideal extractable commitments using stateless tokens. We use some of the ideas employed for the PUF construction, but implement them with different techniques. Indeed, while PUFs are intrinsically unpredictable and even having oracle access to a PUF an unbounded adversary cannot predict the output on a new query, with stateless tokens we do not have such guarantee. Our protocol is secure in the standard token model, where the adversary has no restriction and can send malicious stateful tokens. By plugging such protocol in our compiler, we achieve the

[1] For completeness, we would like to mention that [42] claims an “attack” on such construction. Such “attack" however arises only due to misunderstanding of conventions used to write protocol specifications and does not bear any security threat. The reader can refer to the discussion of [36] (full version of [37]) at Pag. 7, paragraph "On [RvD13]", line 20–40 for more details.

first unconditional UC-secure commitment scheme with stateless tokens. Given that unconditional OT is impossible with stateless tokens, this result completes the picture concerning feasibility of unconditional UC-security in this model.

Related Work. Our compiler can be seen as a generalization of the black-box trapdoor commitment given by Pass and Wee [40] which is secure only in the computational stand-alone setting. Looking ahead to our constructions of extractable commitment, the idea of querying the hardware token with the opening of the commitment was first used by Müller-Quade and Unruh in [33, 34], and later by Chandran et al. in [13]. The construction of [13] builds UC-secure multiple commitments on top of extractable commitments. Their compiler requires computational assumptions, logarithmic number of rounds and crucially uses cryptographic primitives in a non-black box manner.

Dowsley et al. in [17] prove that any statistically hiding commitment obtained from a stateless two-party functionality is also statistically universally composable w.r.t unbounded environment and unbounded simulator. The main difference with our work is that the simulator provided in [17] is not guaranteed to run in polynomial time.

Remark 1. *In the rest of the paper it is assumed that even an unbounded adversary can query the PUF/token only a polynomial number of times. We stress that this is not a restriction of our work but it is a necessary assumption in all previous works achieving unconditional security with PUFs and stateless tokens (see pag.15 of [8] for PUFs, and pag. 5 of [24] for stateless tokens). Indeed, if we allowed the adversary to query the PUF/token on all possible challenges, then she can derive the truth table implemented by the physical device.*

2 Definitions

Notation. We denote the security parameter by n, and the property of a probabilistic algorithm whose number of steps is polynomial in its security parameter, by PPT. We denote by $(v_A, v_B) \leftarrow \langle A(a), B(b)\rangle(x)$ the local outputs of A and B of the random process obtained by having A and B activated with independent random tapes, interacting on common input x and on (private) auxiliary inputs a to A and b to B. When the common input x is the security parameter, we omit it. If A is a probabilistic algorithm we use $v \stackrel{\$}{\leftarrow} A(x)$ to denote the output of A on input x assigned to v. We denote by $\mathsf{view}_A(A(a), B(b))(x)$ the view of A of the interaction with player B, i.e., its values is the transcript $(\gamma_1, \gamma_2, ..., \gamma_t; r)$, where the γ_i's are all the messages exchanged and r is A's coin tosses. We use notation $[n]$ to denote the set $\{1, \dots, n\}$. Let P_1 and P_2 be two parties running protocol (A,B) as sub-routine. When we say that party "P_1 runs $\langle A(\cdot), B(\cdot)\rangle(\cdot)$ with P_2" we always mean that P_1 executes the procedure of party A and P_2 executes the procedure of party B. In the following definitions we assume that the adversary has auxiliary information, and assume that parties are stateful.

2.1 Ideal Extractable Commitment Scheme

We denote by $\mathcal{F}_{\mathsf{aux}}$ an auxiliary set-up functionality accessed by the real world parties (and by the extractor).

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

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

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

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

In this paper the term *ideal* is used to refer to a commitment which is statistically-hiding and statistically-binding.

Definition 2 (Interface Access to an Ideal Functionality $\mathcal{F}_{\mathsf{aux}}$)**.** *Let* $\Pi = (P_1, P_2)$ *be a two-party protocol in the* $\mathcal{F}_{\mathsf{aux}}$*-hybrid model. That is, parties* P_1 *and* P_2 *need to query the ideal functionality* $\mathcal{F}_{\mathsf{aux}}$ *in order to carry out the protocol. An algorithm* M *has* interface access *to the ideal functionality* $\mathcal{F}_{\mathsf{aux}}$ *w.r.t. protocol* Π *if all queries made by either party* P_1 *or* P_2 *to* $\mathcal{F}_{\mathsf{aux}}$ *during the protocol execution are observed (but not answered) by* M*, and* M *has oracle access to* $\mathcal{F}_{\mathsf{aux}}$*. Consequently,* $\mathcal{F}_{\mathsf{aux}}$ *can be a non programmable and non PPT functionality.*

Definition 3 (Ideal Extractable Commitment Scheme in the $\mathcal{F}_{\mathsf{aux}}$ model)**.** $\mathsf{IdealExtCom} = (\mathsf{C}_{\mathsf{ext}}, \mathsf{R}_{\mathsf{ext}})$ *is an* ideal extractable commitment scheme in the $\mathcal{F}_{\mathsf{aux}}$ model *if* $(\mathsf{C}_{\mathsf{ext}}, \mathsf{R}_{\mathsf{ext}})$ *is an ideal commitment and there exists a straight-line strict polynomial-time extractor* E *having* interface access *to* $\mathcal{F}_{\mathsf{aux}}$*, that runs the commitment phase only and outputs a value* $b^\star \in \{0, 1, \perp\}$ *such that, for all malicious committers* C^**, the following properties are satisfied.*

Simulation: *the view generated by the interaction between* E *and* C^* *is identical to the view generated when* C^* *interacts with the honest receiver* $\mathsf{R}_{\mathsf{ext}}$*:* $\mathsf{view}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{C}^*}(\mathsf{C}^*(\mathtt{com}, \cdot), \mathsf{R}_{\mathsf{ext}}(\mathtt{recv})) \equiv \mathsf{view}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{C}^*}(\mathsf{C}^*(\mathtt{com}, \cdot), E)$

Extraction: *let* $\mathbf{c}$ *be a valid transcript of the commitment phase run between* C^* *and* E*. If* E *outputs* $\perp$ *then probability that* C^* *will provide an accepting decommitment is negligible.*

Binding: *if* $b^\star \neq \perp$*, then probability that* C^* *decommits to a bit* $b \neq b^\star$ *is negligible.*

2.2 Physically Uncloneable Functions

Here we recall the definition of PUFs taken from [7]. A Physically Uncloneable Function (PUF) is a noisy physical source of randomness. A PUF is evaluated with a physical stimulus, called the *challenge*, and its physical output, called the *response*, is measured. Because the processes involved are physical, the function implemented by a PUF can not necessarily be modeled as a mathematical function, neither can be considered computable in PPT. Moreover, the output of a PUF is noisy, namely, querying a PUF twice with the same challenge, could yield to different outputs.

A PUF-family $\mathcal{P}$ is a pair of (not necessarily efficient) algorithms Sample and Eval. Algorithm Sample abstracts the PUF fabrication process and works as follows. Given the security parameter in input, it outputs a PUF-index id from the PUF-family satisfying the security property (that we define soon) according to the security parameter. Algorithm Eval abstracts the PUF-evaluation process. On input a challenge s, it evaluates the PUF on s and outputs the response σ. A PUF-family is parametrized by two parameters: the bound on the noisy of the answers d_{noise}, and the size of the range rg. A PUF is assumed to satisfy the bounded noise condition, that is, when running $\mathsf{Eval}(1^n, \mathsf{id}, s)$ twice, the Hamming distance of any two responses σ_1, σ_2 is smaller than $d_{\mathsf{noise}}(n)$. We assume that the challenge space of a PUF is the set of strings of a certain length.

Security Properties. We assume that PUFs enjoy the properties of *uncloneability* and *unpredictability*. Unpredictability is modeled in [7] 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. The following definition automatically implies uncloneability (see [8] pag. 39 for details).

Definition 4 (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 $s \in \{0,1\}^n$ and challenge list $\mathcal{Q} = (s_1, \ldots, s_{\mathtt{poly}(n)})$, one has that, if for all $1 \leq k \leq \mathtt{poly}(n)$ the Hamming distance satisfies $\mathsf{dis}_{\mathsf{ham}}(s, s_k) \geq d_{\mathsf{min}}(n)$, then the average min-entropy satisfies $\tilde{H}_\infty(\mathsf{PUF}(s)|\mathsf{PUF}(\mathcal{Q})) \geq m(n)$, where $\mathsf{PUF}(\mathcal{Q})$ denotes a sequence of random variables $\mathsf{PUF}(s_1), \ldots, \mathsf{PUF}(s_{\mathtt{poly}(n)})$ each one corresponding to an evaluation of the PUF on challenge s_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, querying the PUF twice with the same challenge, one might obtain two distinct responses σ, σ', that are at most d_{noise} apart in hamming distance. Fuzzy extractors of Dodis et al. [21] are applied to the outputs of the PUF, to convert such noisy, high-entropy measurements into *reproducible* randomness. Very roughly, a fuzzy extractor is a pair of efficient randomized algorithms $(\mathsf{FuzGen}, \mathsf{FuzRep})$, and it is applied to PUFs 'responses as follows. FuzGen takes as input an ℓ-bit string, that is the PUF's response σ, and outputs a pair (p, st), where st is a uniformly distributed string, and p is a public helper data string. FuzRep takes as input the PUF's noisy response σ' and the helper data p and generates the very same string st obtained with the original measurement σ.

The security property of fuzzy extractors guarantees that, if the min-entropy of the PUF's responses are greater than a certain parameter m, knowledge of the public data p only, without the measurement σ, does not give any information on the secret value st. The correctness property, guarantees that all pairs of responses σ, σ' that are close enough, i.e., their hamming distance is less then a certain parameter t, will be recovered by FuzRep to the same value st generated by FuzGen. In order to apply fuzzy extractors to PUF's responses it is sufficient to pick an extractor whose parameters match with the parameter of the PUF being used. For formal definitions of fuzzy extractors and PUFs the reader is referred to App. A.1.

Ideal Functionalities for Malicious PUFs and Stateless Tokens. We follow the malicious PUF model introduced in [37]. In this model, the adversary is allowed to create arbitrarily malicious

PUFs. Very informally, a malicious PUF is any physical device that "looks like" a PUF but it implements an arbitrary malicious, possibly stateful, function. Such adversarial behaviour has been modeled in [37] by extending the ideal functionality proposed in [7]. We denote by $\mathcal{F}_{\mathsf{PUF}}$ the ideal functionality for malicious PUF. Its formal description is provided in App. A.3 A stateless token is a wrapper that can be programmed with any arbitrary stateless function. Tokens are modeled by [27, 13] as the ideal functionality $\mathcal{F}_{\mathsf{wrap}}$. $\mathcal{F}_{\mathsf{wrap}}$ is described in App. A.4. Definition of UC-security are provided in App. 10. .

3 The Compiler

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

Extractable Blobs. "Blob" was used in [6] to denote a commitment. In this paper a blob is a *pair* of bit commitments such that the actual bit committed in the blob is the xor of the pair. The representation of a bit as its exclusive-or allows to prove equality of the bits committed in two blobs using commitments as black boxes. Let IdealExtCom be any ideal extractable commitment scheme satisfying Def. 3. If the commitment phase of IdealExtCom is interactive then the blob is the pair of transcripts obtained from the interaction. Procedures to create a blob of a bit b, and to reveal the bit committed in the blob, are the following.

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

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

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

Equality of Blobs. Given the representation of a bit commitment as a blob, a protocol due to Kilian [29] allows to prove that two committed bits (two blobs) are equal, without revealing their values. We build upon this protocol to construct a "simulatable" version, meaning that (given some trapdoor) a simulator can prove equality of two blobs that are *not* equal. Let $\mathbf{B}_i, \mathbf{B}_j$ be two blobs. Let $b_i = (b_i^0 \oplus b_i^1)$ be the bit committed in $\mathbf{B}_i$, and $b_j = (b_j^0 \oplus b_j^1)$ be the bit committed in $\mathbf{B}_j$. Let P denote the prover and V the verifier. In the following protocol P proves to V that $\mathbf{B}_i$ and $\mathbf{B}_j$ are the commitment of the same bit (i.e., $b_i = b_j$).

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

1. V uniformly chooses $e \in \{0, 1\}$ and commits to e using IdealExtCom.
2. P sends $y = b_i^0 \oplus b_j^0$ to V.
3. V reveals e to P.

4. P reveals b_i^e and b_j^e (i.e., P sends decommitments d_i^e, d_j^e to V). V accepts iff $y = b_i^e \oplus b_j^e$.

Protocol BobEquality satisfies the following properties. **Soundness:** if $b_i \neq b_j$, any malicious prover P^* convinces V with probability negligibly close to 1/2, that is the probability of guessing the challenge e. Here we are using the statistically hiding property of the ideal commitment IdealExtCom used to commit e. **Privacy:** If $b_i = b_j$ then after executing the protocol, the view of any verifier V^*, is independent of the actual value of b_i, b_j (given that $\mathbf{B}_i, \mathbf{B}_j$ were secure at the beginning of the protocol). **Simulation:** there exists a straight-line strictly PPT simulator SimFalse such that, for any $(\mathbf{B}_i, \mathbf{B}_j)$ that are not equal (i.e., $b_i \neq b_j$), for any malicious verifier V^*, produces a view that is statistically close to the case in which $(\mathbf{B}_i, \mathbf{B}_j)$ are equal (i.e., $b_i = b_j$) and V^* interacts with the honest P. The above properties are formally proved in Appendix B. Note that the protocol uses blobs in a black-box way. Note also, that a blob can be involved in a single proof only.

3.1 Unconditional UC-secure Commitments from Ideal Extractable Commitments

We construct unconditional UC-secure commitments using *extractable* blobs and protocol BobEquality as building blocks. We want to implement the following idea. The committer sends two blobs of the same bit and proves that they are equal running protocol BobEquality. In the decommitment phase, it opens only one blob (a similar technique is used in [26], where instead the commitment scheme is crucially used in a non black-box way). The simulator extracts the bit of the committer by exploiting the extractability property of blobs. It equivocates by committing to the pair 0, 1 and cheating in the protocol BobEquality. In the opening phase, it then opens the blob corresponding to the correct bit. This idea does not work straight-forwardly since soundness of protocol BobEquality holds only with probability 1/2 and thus a malicious committer can break binding with the same probability. We cannot amplify the soundness by running many proofs on the same pair of blobs, since a blob can be involved in a proof only once. (This is due to the fact that we treat blobs in a black-box manner). Running many proofs among many independent pairs of blobs, and ask the committer to open half of them, is the way to go.

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

Theorem 1. *If* IdealExtCom *is an ideal extractable commitment scheme in the $\mathcal{F}_{\mathsf{aux}}$-hybrid model, then protocol in Fig. 1 is an unconditionally UC-secure bit commitment scheme in the $\mathcal{F}_{\mathsf{aux}}$-hybrid model.*

Proof Sketch. To prove UC-security we have to show a straight-line simulator Sim which correctly simulates the view of the real-world adversary $\mathcal{A}$ and extracts her input. Namely, when simulating the malicious committer in the ideal world, Sim internally runs the real-world adversarial committer $\mathcal{A}$ simulating the honest receiver to her, so to extract the bit committed to by $\mathcal{A}$, and play it in the ideal world. This property is called extractability. When simulating the malicious receiver

Protocol $\mathsf{UCComCompiler}$

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

Commitment Phase

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

Decommitment Phase

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

Figure 1: UCComCompiler: Unconditional UC Commitment from any Ideal Extractable Commitment.

in the ideal world, Sim internally runs the real-world adversarial receiver $\mathcal{A}$ simulating the honest committer to her, without knowing the secret bit to commit to, but in such a way that it can be opened as any bit. This property is called equivocality. In the following, we briefly explain why both properties are achieved.

Straight-line Extractability. It follows from the straight-line extractability and binding of IdealExtCom and from the soundness of protocol BobEquality. Roughly, Sim works as follows. It plays the commitment phase as an honest receiver (and running the straight-line extractor of IdealExtCom having access to $\mathcal{F}_{\mathsf{aux}}$). If all proofs of BobEquality are *successful*, Sim extracts the bits of each consecutive pair of blobs and analyses them as follows. Let $b \in \{0,1\}$. If all extracted pairs of bits are either (b, b) or $(\bar{b}, b)$, (i.e. there are no pairs like $(\bar{b}, \bar{b})$), it follows that, the only bit that $\mathcal{A}$ can successfully decommit to, is b. In this case, Sim plays the bit b in the ideal world. If there is at least a pair (b, b) and a pair $(\bar{b}, \bar{b})$, then $\mathcal{A}$ cannot provide any accepting decommitment (indeed, due to the binding of blobs, $\mathcal{A}$ can only open the bit b from one pair, and the bit $\bar{b}$ from another pair, thus leading the receiver to reject). In this case Sim sends a random bit to the ideal functionality. If all the pairs of blobs are not equal, i.e., all pairs are either $(\bar{b}, b)$ or $(b, \bar{b})$, then $\mathcal{A}$ can later decommit to any bit. In this case the simulator fails in the extraction of the bit committed, and it aborts. Note that, this case happens only when *all* the pairs are not equal. Thus $\mathcal{A}$ was able to cheat in all executions of BobEquality. Due to the soundness of BobEquality, this event happens with probability negligible close to 2^{-n}.

Straight-line Equivocality. It follows from the simulation property of BobEquality. Sim prepares n pairs such that each pair contains blob of 0 and blob of 1, in randomly chosen positions. Then it cheats in all executions of BobEquality, by running the straight-line simulator associated to this protocol. In the decommitment phase, after having received the bit to decommit to, for each pair, Sim reveals the blob corresponding to the correct bit.

Note that, in both cases Sim crucially uses the extractor associated to IdealExtCom, that in turn uses the access to $\mathcal{F}_{\mathsf{aux}}$. The formal proof of Theorem 1 can be found in App. C.

In Section 4 we show an implementation of IdealExtCom with malicious PUFs, while in Section 5, we show how to implement IdealExtCom using stateless token. By plugging such implementations in protocol UCComCompiler we obtain the first unconditional UC-secure commitment scheme with malicious PUFs (namely, in the $\mathcal{F}_{\mathsf{PUF}}$-hybrid model), and stateless tokens (namely, in the $\mathcal{F}_{\mathsf{wrap}}$-hybrid model).

4 Ideal Extractable Commitment from (Malicious) PUFs

In this section we show a construction of ideal extractable commitment in the $\mathcal{F}_{\mathsf{PUF}}$ model. Our construction builds upon the ideal commitment scheme presented in [37]. We denote this protocol by CPuf. For simplicity, in the informal description of the protocol we omit mentioning the use of fuzzy extractors and the formalism for invoking the $\mathcal{F}_{\mathsf{PUF}}$ functionality. Such details are provided in the formal descriptions (Fig. 8 and Fig. 2)..

Ideal Commitment Scheme in the $\mathcal{F}_{\mathsf{PUF}}$ Model (from [37]). The idea behind the protocol of [37], that we denote by $\mathsf{CPuf} = (\mathsf{C}_{\mathsf{CPuf}}, \mathsf{R}_{\mathsf{CPuf}})$, is to turn Naor's commitment scheme [35] [2] which is statistically binding but only computationally hiding, into statistically hiding and binding, by replacing the PRG with a (possibly *malicious*) PUF. Roughly, protocol CPuf goes as follows. At the beginning of the protocol, the committer creates a PUF, that we denote by $\mathcal{P}_S$. It preliminary queries $\mathcal{P}_S$ with a random string s (of n bits) to obtain the response σ_S (of $rg(3n)$ bits, where rg is the PUF's range) and finally sends the PUF $\mathcal{P}_S$ to the receiver. After receiving the PUF, the receiver sends a random string r (i.e., the first round of Naor's commitment) to the committer. To commit to a bit b, the committer sends $\mathbf{c} = \sigma_S \oplus (r \wedge b^{|r|})$ to the receiver. In the decommitment phase, the committer sends (b, s) to the receiver, who checks the commitment by querying $\mathcal{P}_S$ with s. Hiding intuitively follows from the fact that a fuzzy extractor applied to the PUF-response σ_S yields to a uniformly distributed value. Thus, commitment of 1 ,i.e., $\mathbf{c} = \sigma_S \oplus r$ and commitment of 0, i.e., $\mathbf{c} = \sigma_S$, are statistically close. Binding follows the same argument of Naor's scheme and is based on the expansion property of the PUF where the challenge is n bits and the response is $rg(3n)$ bits (for more details on the proof the reader is referred to [36]). The formal description of CPuf is provided in Fig. 8 in App. D.

Our Ideal *Extractable* Commitment Scheme in the $\mathcal{F}_{\mathsf{PUF}}$ Model. We transform CPuf into a *straight-line extractable* commitment using the following technique. We introduce a new PUF $\mathcal{P}_R$, sent by the receiver to the committer at the beginning of the protocol. Then we force the committer to query the PUF $\mathcal{P}_R$ with the opening of the commitment the same idea is used in [33, 13])computed running CPuf. An opening of protocol CPuf is the value σ_S[3]. This allows the extractor, who

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

[3]In the actual implementation we require the committer to query $\mathcal{P}_R$ with the output of the fuzzy extractor st_S, i.e., $(st_S, p_S) \leftarrow \mathsf{FuzGen}(\sigma_S)$.

has access to the interface of $\mathcal{F}_{\mathsf{PUF}}$, to extract the opening. Note that extractability must hold against a malicious committer, in which case the token $\mathcal{P}_R$ sent by the receiver is honest, therefore the extractor is allowed to intercept such queries. The idea is that, from the transcript of the commitment (i.e., the value $\mathbf{c} = \sigma_S \oplus (r \wedge b)$) and the queries made to $\mathcal{P}_R$, (the value σ_S) the bit committed if fully determined. [4].

How can we force the committer to query $\mathcal{P}_R$ with the correct opening? We require that it commits to the answer σ_R obtained by $\mathcal{P}_R$, using again protocol CPuf. Why the committer cannot send directly the answer σ_R? Because σ_R could be the output of a malicious PUF, and leak information about the query made by the committer. Thus, in the commitment phase, the committer runs two instances of CPuf. One instance, that we call ComBit, is run to commit to the secret bit b. The other instance, that we call ComResp, is run to commit to the response of PUF $\mathcal{P}_R$, queried with the opening of ComBit. In the decommitment phase, the receiver gets $\mathcal{P}_R$ back, along with the openings of both the bit and the PUF-response. Then it queries $\mathcal{P}_R$ with the opening of ComBit, and checks if the response is consistent with the string committed in ComResp. Due to the unpredictability of PUFs, the committer cannot guess the output of $\mathcal{P}_R$ on the string σ_S without querying it. Due to the statistically binding of CPuf, the committer cannot postpone querying the PUF in the decommitment phase. Thus, if the committer will provide a valid decommitment, the extractor would have observed the opening already in the commitment phase with all but negligible probability.

However, there is one caveat. The unpredictability of PUFs is guaranteed only for queries that are sufficiently apart from each other. Which means that, given a challenge/response pair (c, r), the response on any strings c' that is "close" in hamming distance to c ("close" means that $\mathsf{dis}_{\mathsf{ham}}(c, c') \leq d_{\mathsf{min}}$), could be predictable. Consequently, a malicious committer could query the PUF with a string that is only "close" to the opening. Then, given the answer to such a query, she could predict the answer to the actual opening, *without* querying the PUF. In this case, the extractor cannot determine which is the opening, since it cannot try all possible strings that are "close" to queries made by the malicious committer. Thus the extraction fails. At the same time, the malicious committer did not violate the unpredictability property of PUFs, since it predicted a value that is "too close" to the one already observed.

We overcome this problem by using Error Correcting Codes, in short ECC (see Def. 6). The property of an ECC with distance parameter dis, is that any pair of strings having hamming distance dis, decodes to a unique string. Therefore, we modify the previous approach asking the committer to query PUF $\mathcal{P}_R$ with the *encoding* of the opening, i.e., $\mathsf{Encode}(\sigma_S)$. In this way, all queries that are "too close" in hamming distance decode to the same opening, and the previous attack is defeated. Informally, hiding and biding follow from hiding and binding of CPuf. Indeed, protocol ExtPuf basically consists in running two instances of CPuf in parallel. Extractability follows from the statistically biding of CPuf, the unpredictability of $\mathcal{P}_R$ and the Minimum Distance Property of ECC. The formal description of the above protocol, that we denote by $\mathsf{ExtPuf} = (\mathsf{C}_{\mathsf{ExtPuf}}, \mathsf{R}_{\mathsf{ExtPuf}})$, is shown in Fig. 2.

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

Protocol ExtPuf

$\mathsf{ECC} = (\mathsf{Encode}, \mathsf{Decode})$ is a $(N, L, d^1_{\mathsf{min}})$ error correcting code, where $L = \ell = 3n$. $\mathcal{F}_{\mathsf{PUF}}$ is parameterized with a PUF family $\mathcal{P}^1{=}(rg^1, d^1_{\mathsf{noise}}, d^1_{\mathsf{min}}, m^1)$, with challenge size L. $(\mathsf{FuzGen}^1, \mathsf{FuzRep}^1)$ is a $(m^1, \ell^1, t^1, \epsilon^1)$-fuzzy extractor of appropriate matching parameters. Protocol $\mathsf{CPuf} = (\mathsf{C}_{\mathsf{CPuf}}, \mathsf{R}_{\mathsf{CPuf}})$ (depicted in Fig. 8) is run as sub-routine. $\mathcal{P}_S$ is the sid used to denote the PUF created by the committer in CPuf. $\mathcal{P}_R$ is the sid of the PUF created by the receiver. Committer's Input: $b \in \{0, 1\}$.

Commitment Phase

1. Receiver $\mathsf{R}_{\mathsf{ExtPuf}}$: create a PUF sending $(\mathsf{init}_{\mathsf{PUF}}, \mathcal{P}_R, \mathtt{normal}, \mathsf{R}_{\mathsf{ExtPuf}})$ to $\mathcal{F}_{\mathsf{PUF}}$ and then handover it to $\mathsf{C}_{\mathsf{ExtPuf}}$ sending $(\mathsf{handover}_{\mathsf{PUF}}, \mathcal{P}_R, \mathsf{R}_{\mathsf{ExtPuf}}, \mathsf{C}_{\mathsf{ExtPuf}})$ to $\mathcal{F}_{\mathsf{PUF}}$.
2. **Commitment of the Secret Bit:** ComBit.
   $\mathsf{C}_{\mathsf{ExtPuf}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtPuf}}$: run $\langle \mathsf{C}_{\mathsf{CPuf}}(\mathtt{com}, b), \mathsf{R}_{\mathsf{CPuf}}(\mathtt{com})\rangle$ so that $\mathsf{C}_{\mathsf{ExtPuf}}$ commits to bit b. Let $(st_S, p_S) \leftarrow \mathsf{FuzGen}(\sigma_S)$ be the value obtained by $\mathsf{C}_{\mathsf{ExtPuf}}$, after applying the fuzzy extractor to the answer obtained from its own PUF $\mathcal{P}_S$ when running protocol ComBit.
3. Committer $\mathsf{C}_{\mathsf{ExtPuf}}$: Send $(\mathsf{eval}_{\mathsf{PUF}}, \mathcal{P}_R, \mathsf{C}_{\mathsf{ExtPuf}}, \mathsf{Encode}(st_S))$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain response $(\mathsf{response}_{\mathsf{PUF}}, \mathcal{P}_R, \mathsf{Encode}(st_S), \sigma_R)$. If $\sigma_R = \perp$ (i.e., PUF $\mathcal{P}_R$ aborts), set $\sigma_R \leftarrow 0$. Compute $(st_R, p_R) \leftarrow \mathsf{FuzGen}^1(\sigma_R)$.
4. **Commitment of $\mathcal{P}_R$'s Response:** $\mathsf{ComResp}$.
   $\mathsf{C}_{\mathsf{ExtPuf}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtPuf}}$: run $\langle \mathsf{C}_{\mathsf{CPuf}}(\mathtt{com}, st_R||p_R), \mathsf{R}_{\mathsf{CPuf}}(\mathtt{com})\rangle$ so that $\mathsf{C}_{\mathsf{ExtPuf}}$ commits to the string $st_R||p_R$.

Decommitment Phase

1. $\mathsf{C}_{\mathsf{ExtPuf}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtPuf}}$: run $\langle \mathsf{C}_{\mathsf{CPuf}}(\mathtt{open}, b), \mathsf{R}_{\mathsf{CPuf}}(\mathtt{open})\rangle$ and $\langle \mathsf{C}_{\mathsf{CPuf}}(\mathtt{open}, st_R||p_R), \mathsf{R}_{\mathsf{CPuf}}(\mathtt{open})\rangle$.
2. Committer $\mathsf{C}_{\mathsf{ExtPuf}}$: handover PUF $\mathcal{P}_R$ to $\mathsf{R}_{\mathsf{ExtPuf}}$ by sending $(\mathsf{handover}_{\mathsf{PUF}}, \mathcal{P}_R, \mathsf{C}_{\mathsf{CPuf}}\mathsf{R}_{\mathsf{ExtPuf}})$ to $\mathcal{F}_{\mathsf{PUF}}$.
3. Receiver $\mathsf{R}_{\mathsf{ExtPuf}}$: If both decommitments are successfully completed, then $\mathsf{R}_{\mathsf{ExtPuf}}$ gets the bit b' along with the opening st'_S for ComBit and string $st'_R||p'_R$ for $\mathsf{ComResp}$.
   Check validity of st'_R: send $(\mathsf{eval}_{\mathsf{PUF}}, \mathcal{P}_R, \mathsf{R}_{\mathsf{ExtPuf}}, \mathsf{Encode}(st'_S))$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain σ'_R. Compute $st''_R \leftarrow \mathsf{FuzRep}^1(\sigma'_R, p'_R)$. If $st''_R = st'_R$, then accept and output b. Else, reject.

Figure 2: ExtPuf: Ideal **Extractable** Commitment in the $\mathcal{F}_{\mathsf{PUF}}$ model.

PUF parameters of ExtPuf **for Bit Commitment.** ExtPuf requires two PUFs $\mathcal{P}_S, \mathcal{P}_R$. $\mathcal{P}_S$ is sent by the committer to the receiver when executing CPuf as sub-protocol. CPuf is used to commit to a bit b and to the response of PUF $\mathcal{P}_R$ which is of size $K = |(st_R||p_R)|$. $\mathcal{P}_S$ has challenge size n and range size $rg(K\ell)$ (with $\ell = 3n$). $\mathcal{P}_R$ is sent by the receiver and is used for extractability only. $\mathcal{P}_R$ has challenge size L and range size $rg^1(L)$. Note that parameters of $\mathcal{P}_R$ are independent of the number of bits that are committed.

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

On round complexity of ExtPuf. For simplicity in Fig. 2 we describe the interaction between $\mathsf{C}_{\mathsf{ExtPuf}}$ and $\mathsf{R}_{\mathsf{ExtPuf}}$ using several rounds. However, we stress that the exchange of the PUF can be done once at the beginning of the protocol, and that except from the PUF transfer, the commitment phase requires only three rounds. The decommitment is non-interactive and requires another PUF transfer.

Theorem 2. *If* CPuf *is an Ideal Commitment in the* $\mathcal{F}_{\mathsf{PUF}}$*-model, then* ExtPuf *is an Ideal* Extractable *Commitment in the* $\mathcal{F}_{\mathsf{PUF}}$ *model.*

The proof of this theorem is provided in Appendix E.1.

5 Ideal Extractable Commitments from Stateless Tokens

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

Ideal Commitment Scheme in the $\mathcal{F}_{\mathsf{wrap}}$ Model. Similarly to the construction with PUFs, we implement Naor's commitment scheme by replacing the PRG with a stateless token.

In the construction with PUFs, the PRG was replaced with a PUF that is inherently unpredictable. Indeed, by assumption, even after observing a polynomial number of challenge/response pairs, a malicious receiver cannot predict the output of the PUF on any new (sufficiently far apart) challenge. In this case, hiding breaks only if the receiver guesses the challenge used by the committer, which happens only with negligible probability. Hence, hiding holds unconditionally. Now, we want to achieve statistically hiding using *stateless* token. The problem here is that we do not have unpredictability for free (as it happens with PUFs). Thus, we have to program the stateless token with a function that is, somehow, unconditionally unpredictable. Clearly, we cannot construct a token that implements a PRG. Indeed, after observing a few pairs of input/output, an unbounded receiver can extract the seed, compute all possible outputs, and break hiding. We use a point function following [24] . A point function f is a function that outputs always zero, except in a particular point x, in which it outputs a value y. Formally, $f : \{0,1\}^n \rightarrow \{0,1\}^m$ such that $f(x) = y$ and $f(x') = 0$, for all $x' \neq x$.

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

The statistically binding property follows from the same arguments of Naor's scheme. The token is sent away before committer can see r. Thus, since x is only n bits, information theoretically the committer cannot instruct a malicious token to output y' adaptively on x. Thus, for any malicious possibly *stateful* token, binding is preserved. The statistically hiding property holds due to the fact that x is secret. A malicious receiver can query the token with any polynomial number of values x'. But, whp she will miss x, and thus she will obtain always 0. Such an answer does not help her to predict y. The only way to obtain y is to predict x. This happens with probability 2^{-n}.

The above protocol is denoted by CTok and is formally described in Fig. 3. We stress that, this is the first construction of unconditional commitment scheme that is secure even against malicious *stateful* tokens. The previous construction of unconditional commitment scheme of [24] is secure as long as the malicious token is stateless (i.e., it assumes that the adversary cannot create stateful tokens). Furthermore, our constructions requires only one stateless token, while construction of [24] requires a number of tokens that depends on the security parameter.

From Bit Commitment to String Commitment. To commit to a ℓ-bit string using one stateless token only is sufficient to embed ℓ pairs $(x_1, y_1), \ldots, (x_\ell, y_\ell)$ in the token $\mathcal{T}_\mathsf{C}$ and to require that for each i, $x_i \in \{0,1\}^n$ and $y_i \in \{0,1\}^{3\ell n}$. Namely, $\mathcal{T}_\mathsf{C}$ grows linearly with the size of the string to be committed. Then, execute protocol CTok for each bit of the string in parallel. The receiver accepts the string iff all bit commitments are accepting.

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

Commitmen Phase

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

Decommitment Phase

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

Figure 3: CTok: Ideal Commitments in the $\mathcal{F}_\mathsf{wrap}$ model.

Ideal Extractable Commitment in the $\mathcal{F}_\mathsf{wrap}$ model. Extractability is achieved as before. The receiver sends a token $\mathcal{T}_\mathsf{R}$ to the committer. The committer is required to query $\mathcal{T}_\mathsf{R}$ with the opening of the commitment (namely, the value y) and then commit to the token's response. In the decommitment phase, the committer opens both the commitment of the bit and of the token's response. The receiver then checks that the latter value corresponds to the response of $\mathcal{T}_\mathsf{R}$ on input the opening of the commitment of the bit. Note that here the receiver can check the validity of the token's response without physically possessing the token.

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

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

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

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

As final step, the committer commits to the token's response σ_tok, using the scheme CTok. (If the token of the receiver aborts, the committer sets σ_tok to the zero string). In the decommitment phase, the receiver first checks the validity of both commitments (commitment of the bit, commitment of the answer σ_tok). Then, given the opening of the bit, it checks that σ_tok is a valid MAC computed under key k_tok on such opening.

Binding follows from the binding of CTok and the unforgeability of MAC. Hiding still follows from the hiding of CTok. Indeed, the answer of $\mathcal{T}_\mathsf{R}$ sent by the malicious receiver, is not forwarded to the receiver, but is committed using the ideal commitment CTok. Furthermore, if $\mathcal{T}_\mathsf{R}$ selectivly aborts, the committer does not halt but it continues committing to the zero-string. The receiver will get its token's answer in clear only in the decommitment phase when the bit has been already revealed. The formal description of the above protocol, that we denote by ExtTok, is shown in Fig. 4.

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

Commitment Phase

1. Receiver $\mathsf{R}_{\mathsf{ExtTok}}$: pick MAC-keys: $k_{\mathsf{rec}}, k_{\mathsf{tok}}$. Create token $\mathcal{T}_{\mathsf{R}}$ implementing the following functionality. On input a tuple $(r||\mathbf{c}, \sigma_{\mathsf{rec}}, y)$: if $\mathsf{Vrfy}(k_{\mathsf{rec}}, r||\mathbf{c}, \sigma_{\mathsf{rec}}) = 1$ and ($\mathbf{c} = y$ OR $\mathbf{c} = y \oplus r$) then output $\sigma_{\mathsf{tok}} = \mathsf{Mac}(k_{\mathsf{tok}}, y)$ else output $\perp$.
(Formally, $\mathsf{R}_{\mathsf{ExtTok}}$ sends (create, sid, $\mathsf{R}_{\mathsf{ExtTok}}$, $\mathsf{C}_{\mathsf{ExtTok}}$, $\mathcal{T}_{\mathsf{R}}$) to $\mathcal{F}_{\mathsf{wrap}}$). Send $\mathcal{T}_{\mathsf{R}}$ to the sender $\mathsf{C}_{\mathsf{ExtTok}}$.
Commitment of the Secret Bit: ComBit.
2. $\mathsf{C}_{\mathsf{ExtTok}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtTok}}$: run $\langle \mathsf{C}_{\mathsf{CTok}}(\mathtt{com}, b), \mathsf{R}_{\mathsf{CTok}}(\mathtt{com})\rangle$ so that $\mathsf{C}_{\mathsf{CTok}}$ commits to bit b. Let $(r, \mathbf{c})$ be the transcript of such commitment phase. Let y be the opening of $\mathbf{c}$.
3. Receiver $\mathsf{R}_{\mathsf{ExtTok}}$: compute $\sigma_{\mathsf{rec}} \leftarrow \mathsf{Mac}(k_{\mathsf{rec}}, r||\mathbf{c})$. Send σ_{rec} to Committer $\mathsf{C}_{\mathsf{ExtTok}}$.
4. Committer $\mathsf{C}_{\mathsf{ExtTok}}$: query $\mathcal{T}_{\mathsf{R}}$ with $q = (r||\mathbf{c}, \sigma_{\mathsf{rec}}, y)$ (i.e., send $(\mathsf{run}, \mathsf{sid}, \mathsf{C}_{\mathsf{ExtTok}}, \mathcal{T}_{\mathsf{R}}, q)$ to $\mathcal{F}_{\mathsf{wrap}}$) and obtain σ_{tok}. If token $\mathcal{T}_{\mathsf{R}}$ aborts, set $\sigma_{\mathsf{tok}} = 0^n$.
Commitment of $\mathcal{T}_{\mathsf{R}}$'s Response: $\mathsf{ComResp}$.
$\mathsf{C}_{\mathsf{ExtTok}} \Leftrightarrow \mathsf{R}_{\mathsf{ExtTok}}$: run $\langle \mathsf{C}_{\mathsf{CTok}}(\mathtt{com}, \sigma_{\mathsf{tok}}), \mathsf{R}_{\mathsf{CTok}}(\mathtt{com})\rangle$ so that $\mathsf{C}_{\mathsf{ExtTok}}$ commits to the response σ_{tok} received from $\mathcal{T}_{\mathsf{R}}$.

Decommitment Phase

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

Figure 4: ExtTok: Ideal **Extractable** Commitment in the $\mathcal{F}_{\mathsf{wrap}}$ model.

Theorem 3. *Protocol* ExtTok *is an ideal* extractable *commitment in the* $\mathcal{F}_{\mathsf{wrap}}$ *model.*

The proof of Theorem 3 is provided in Appendix E.2.

Acknowledgments

The second author thanks Dominik Scheder for very interesting discussions on Azuma's inequality, and Akshay Wadia for suggesting a simplification in the compiler. The same author thanks Ivan Visconti and Rafail Ostrovsky for valuable discussions.

References

[1] Armknecht, F., Maes, R., Sadeghi, A.R., Standaert, F.X., Wachsmann, C.: A formalization of the security features of physical functions. In: IEEE Symposium on Security and Privacy. pp. 397–412. IEEE Computer Society (2011)

[2] Armknecht, F., Maes, R., Sadeghi, A.R., Sunar, B., Tuyls, P.: Memory leakage-resilient encryption based on physically unclonable functions. In: Matsui, M. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 5912, pp. 685–702. Springer (2009)

[3] Barak, B., Canetti, R., Nielsen, J.B., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: Foundations of Computer Science (FOCS'04). pp. 394–403 (2004)

[4] BenOr, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC. pp. 1–10 (1988)

[5] Boit, C., Helfmeier, C., Nedospasaov, D., Seifert, J.P.: Cloning physically unclonable functions. IEEE HOST (2013)

[6] Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988)

[7] Brzuska, C., Fischlin, M., Schröder, H., Katzenbeisser, S.: Physically uncloneable functions in the universal composition framework. In: Rogaway, P. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6841, pp. 51–70. Springer (2011)

[8] Brzuska, C., Fischlin, M., Schröder, H., Katzenbeisser, S.: Physically uncloneable functions in the universal composition framework. IACR Cryptology ePrint Archive 2011, 681 (2011)

[9] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Foundations of Computer Science (FOCS'01). pp. 136–145 (2001)

[10] Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: TCC. pp. 61–85 (2007)

[11] Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Biham, E. (ed.) Advances in Cryptology – EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pp. 68–86. Springer, Berlin, Germany, Warsaw, Poland (May 4–8, 2003)

[12] Canetti, R., Lindell, Y., Ostrovsky, R., Sahai., A.: Universally composable two-party and multi-party secure computation. In: 34th Annual ACM Symposium on Theory of Computing. pp. 494–503. Lecture Notes in Computer Science, ACM Press, Montréal, Québec, Canada (May 19–21, 2002)

[13] Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 545–562. Springer, Berlin, Germany, Istanbul, Turkey (2008)

[14] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC. pp. 11–19 (1988)

[15] Choi, S.G., Katz, J., Schröder, D., Yerukhimovich, A., Zhou, H.S.: (efficient) universally composable two-party computation using a minimal number of stateless tokens. IACR Cryptology ePrint Archive 2011, 689 (2011)

[16] Crépeau, C., Kilian, J.: Achieving oblivious transfer using weakened security assumptions (extended abstract). In: FOCS. pp. 42–52. IEEE Computer Society (1988)

[17] Rafael Dowsley R., Jeroen van de Graaf and Jörn Müller-Quade and Anderson C. A. Nascimento: On the Composability of Statistically Secure Bit Commitments. IACR Cryptology ePrint Archive 2008, 457 (2008)

[18] Damgård, I., Kilian, J., Salvail, L.: On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In: EUROCRYPT. pp. 56–73 (1999)

[19] Damgård, I., Scafuro, A.: Unconditionally secure and universally composable commitments from physical assumptions. IACR Cryptology ePrint Archive 2013, 108 (2013)

[20] van Dijk, M., Rührmair, U.: Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results. IACR Cryptology ePrint Archive 2012, 228 (2012)

[21] Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM J. Comput. 38(1), 97–139 (2008)

[22] Döttling, N., Kraschewski, D., Müller-Quade, J.: Unconditional and composable security using a single stateful tamper-proof hardware token. In: Ishai, Y. (ed.) TCC. Lecture Notes in Computer Science, vol. 6597, pp. 164–181. Springer (2011)

[23] Döttling, N., Kraschewski, D., Müller-Quade, J.: David & goliath oblivious affine function evaluation - asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token. IACR Cryptology ePrint Archive 2012, 135 (2012)

[24] Goyal, V., Ishai, Y., Mahmoody, M., Sahai, A.: Interactive locking, zero-knowledge pcps, and unconditional cryptography. In: Advances in Cryptology – CRYPTO 2010. pp. 173–190. Lecture Notes in Computer Science, Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 2010)

[25] Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Micciancio, D. (ed.) TCC 2010: 7th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 5978, pp. 308–326. Springer, Berlin, Germany, Zurich, Switzerland (Feb 9–11, 2010)

[26] Hofheinz, D.: Possibility and impossibility results for selective decommitments. J. Cryptology 24(3), 470–516 (2011)

[27] Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) Advances in Cryptology – EUROCRYPT 2007. Lecture Notes in Computer Science, vol. 4515, pp. 115–128. Barcelona, Spain (May 20–24, 2007)

[28] Katzenbeisser, S., Koccabas, Ü., Rozic, V., Sadeghi, A.R., Verbauwhede, I., Wachsmann, C.: Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In: CHES. pp. 283–301 (2012)

[29] Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732. STOC '92, ACM, New York, NY, USA (1992), http://doi.acm.org/10.1145/129712.129782

[30] Kolesnikov, V.: Truly efficient string oblivious transfer using resettable tamper-proof tokens. In: Micciancio, D. (ed.) TCC 2010: 7th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 5978, pp. 327–342. Springer, Berlin, Germany, Zurich, Switzerland (Feb 9–11, 2010)

[31] Maes, R., Verbauwhede, I.: Physically unclonable functions: A study on the state of the art and future research directions. In: Sadeghi, A.R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security, pp. 3–37. Information Security and Cryptography, Springer Berlin Heidelberg (2010)

[32] Moran, T., Segev, G.: David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 527–544. Springer, Berlin, Germany, Istanbul, Turkey (Apr 13–17, 2008)

[33] Müller-Quade, J., Unruh, D.: Long-term security and universal composability. In: TCC. pp. 41–60 (2007)

[34] Müller-Quade, J., Unruh, D.: Long-term security and universal composability. J. Cryptology 23(4), 594–671 (2010)

[35] Naor, M.: Bit commitment using pseudo-randomness. In: CRYPTO. pp. 128–136 (1989)

[36] Ostrovsky, R., Scafuro, A., Visconti, I., Wadia, A.: Universally composable secure computation with (malicious) physically uncloneable functions. IACR Cryptology ePrint Archive 2012, 143 (2012)

[37] Ostrovsky, R., Scafuro, A., Visconti, I., Wadia, A.: Universally composable secure computation with (malicious) physically uncloneable functions. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 702–718. Springer (2013)

[38] Pappu, R.S., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science 297, 2026–2030 (2002)

[39] Pappu, R.S.: Physical One-Way Functions. Ph.D. thesis, MIT (2001)

[40] Pass, R., Wee, H.: Black-box constructions of two-party protocols from one-way functions. In: Reingold, O. (ed.) TCC. Lecture Notes in Computer Science, vol. 5444, pp. 403–418. Springer (2009)

[41] Rührmair, U.: Oblivious transfer based on physical unclonable functions. In: Acquisti, A., Smith, S.W., Sadeghi, A.R. (eds.) TRUST. Lecture Notes in Computer Science, vol. 6101, pp. 430–440. Springer (2010)

[42] Rührmair, U., van Dijk, M.: Pufs in security protocols: Attack models and security evaluations. In: IEEE Symposium on Security and Privacy (2013)

[43] Sadeghi, A.R., Visconti, I., Wachsmann, C.: Enhancing rfid security and privacy by physically unclonable functions. In: Sadeghi, A.R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security, pp. 281–305. Information Security and Cryptography, Springer Berlin Heidelberg (2010)

A More Definitions

For two random variables X and Y with supports in $\{0,1\}^n$, the *statistical difference* between X and Y, denoted by $SD(X,Y)$, is defined as, $SD(X,Y) = \frac{1}{2}\sum_{z\in\{0,1\}^n} |\Pr[X=z] - \Pr[Y=z]|$. A function ϵ is negligible in n (or just negligible) if for every polynomial $p(\cdot)$ there exists a value $n_0 \in \mathbb{N}$ such that for all $n > n_0$ it holds that $\epsilon(n) < 1/p(n)$.

Indistinguishability. Let $\mathcal{W}$ be a set of strings. An *ensemble* of random variables $X = \{X_w\}_{w\in\mathcal{W}}$ is a sequence of random variables indexed by elements of $\mathcal{W}$.

Definition 5. *Two ensembles of random variables $X = \{X_w\}_{w\in\mathcal{W}}$ and $Y = \{Y_w\}_{w\in\mathcal{W}}$ are* statistically indistinguishable*, i.e., $\{X_w\}_{w\in\mathcal{W}} \overset{\mathsf{s}}{\equiv} \{Y_w\}_{w\in\mathcal{W}}$ if for any distinguisher D there exists a negligible function ϵ such that*

$$\big|\Pr[\alpha \leftarrow X_w : D(w,\alpha)=1] - \Pr[\alpha \leftarrow Y_w : D(w,\alpha)=1]\big| < \epsilon(w).$$

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

Definition 6 (Error correcting code)**.** *An (N, L, dis)-Error Correcting Code (*ECC*), is a tuple of algorithms* $(\mathsf{Encode}, \mathsf{Decode})$ *where* $\mathsf{Encode} : \{0,1\}^N \rightarrow \{0,1\}^L$ *and* $\mathsf{Decode} : \{0,1\}^L \rightarrow \{0,1\}^N$ *satisfy the following properties:*

- *Efficiency:* $\mathsf{Encode}, \mathsf{Decode}$ *are deterministic polynomial time algorithms;*
- *Minimum Distance:* $\forall m_1, m_2 \in \{0,1\}^N, \mathsf{dis}_{\mathsf{ham}}(\mathsf{Encode}(m_1), \mathsf{Encode}(m_2)) \geq \mathsf{dis}$*;*
- *Correct Decoding:* $\forall m, \mathsf{cd} = \mathsf{Encode}(m)$*,* $\forall \mathsf{cd}' \in \{0,1\}^L$ *such that* $\mathsf{dis}_{\mathsf{ham}}(\mathsf{cd}, \mathsf{cd}') \leq \lfloor \frac{\mathsf{dis}}{2} \rfloor$ *it holds that* $\mathsf{Decode}(c') = m$*.*

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

A.1 More Definitions for PUFs

We follow the definition of PUF provided in [7].

Definition 7 (Physically Uncloneable Functions). *Let rg denote the size of the range 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 $s \in \{0,1\}^n$, $\mathcal{D}_{\mathsf{id}}$ contains a distribution $\mathcal{D}_{\mathsf{id}}(s)$ on $\{0,1\}^{rg(n)}$. $\mathcal{D}_{\mathsf{id}}$ is not necessarily an efficiently samplable distribution.*

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

Bounded Noise. *For all indexes* $\mathsf{id} \in \mathcal{I}_n$, *for all challenges $s \in \{0,1\}^n$, when running* $\mathsf{Eval}(1^n, \mathsf{id}, s)$ *twice, the Hamming distance of any two responses σ_1, σ_2 is smaller than $d_{\mathsf{noise}}(n)$.*

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

Definition 8 (Average min-entropy). *The average min-entropy of the measurement* $\mathsf{PUF}(s)$ *conditioned on the measurements of challenges $\mathcal{Q} = (s_1, \ldots, s_{\mathtt{poly}(n)})$ is defined by:*

$$\begin{aligned}\tilde{H}_\infty(\mathsf{PUF}(s)|\mathsf{PUF}(\mathcal{Q})) &= -log\big(\mathbb{E}_{\sigma_i \leftarrow \mathsf{PUF}(s_i)}[\max_a \Pr\big[\,\mathsf{PUF}(s) = \sigma|\sigma_1 = \mathsf{PUF}(s_1), \ldots, \sigma_{\mathtt{poly}(n)} = \mathsf{PUF}(s_{\mathtt{poly}(n)})\,\big]\,\big) \\ &= -log\big(\mathbb{E}_{\sigma_i \leftarrow \mathsf{PUF}(s_i)}[2^{H_\infty(\mathsf{PUF}(s)=\sigma|\sigma_1=\mathsf{PUF}(s_1),\ldots,\sigma_{\mathtt{poly}(n)}=\mathsf{PUF}(s_{\mathtt{poly}(n)}))}\big)\end{aligned}$$

where the probability is taken over the choice of id *from the PUF-family and the choice of possible PUF responses on challenge s. The term* $\mathsf{PUF}(\mathcal{Q})$ *denotes a sequence of random variables* $\mathsf{PUF}(s_1), \ldots, \mathsf{PUF}(s_{\mathtt{poly}(n)})$ *each corresponding to an evaluation of the PUF on challenge s_k.*

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

Definition 9 (Fuzzy Extractors). *Let* dis *be a distance function for metric space $\mathcal{M}$. A (m, ℓ, t, ϵ)-fuzzy extractor is a pair of efficient randomized algorithms* (FuzGen, 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., $SD((st, p), (U_\ell, p)) \leq \epsilon$.*

Fuzzy Extractors Applied to PUF's output. Given a $(rg(n), d_{\mathsf{noise}}(n), d_{\mathsf{min}}(n), m(n))$-PUF family with $d_{\mathsf{min}} = o(n/\log n)$, a *matching* fuzzy extractor has the following parameters: $\ell(n) = n$, $t(n) = d_{\mathsf{noise}}(n)$, and ϵ is a negligible function in 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 $s_1, \ldots, s_{p(n)}$, the probability that a randomly chosen challenge is within distance smaller than d_{min} with any s_k is negligible.

Extraction Independence. For all challenges $s_1, \ldots, s_{p(n)}$, and for a challenge s such that $\mathsf{dis}(s, s_k) > d_{\mathsf{min}}$ for $1 \leq k \leq p(n)$, it holds that the PUF evaluation on s and subsequent application of FuzGen yields an almost uniform value st even if p is observed.

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

A.2 Ideal Functionalities and the UC framework

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

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

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

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

In the universally composable framework [9], the distinguisher of the views is the environment $\mathcal{Z}$. $\mathcal{Z}$ has the power of choosing the inputs of all the parties and guide the actions of the adversary $\mathcal{A}$ (scheduling messages, corrupting parties), who will act just as proxy overall the execution. Let $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$ be the distribution ensemble that describes the environment's output in the ideal world process, and $\mathsf{REAL}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{Z}}$ the distribution of the environment's output in the real world process in the $\mathcal{G}$-hybrid model.

Definition 10 (Information theoretically UC-security)**.** *Let $\mathcal{F}$ be an ideal functionality, and Π be a PPT protocol. We say Π **realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model** if for any hybrid-model static adversary $\mathcal{A}$, there exists an ideal world expected PPT adversary Sim such that for every environment $\mathcal{Z}$, for all auxiliary information to $z \in \{0,1\}^*$ to $\mathcal{Z}$, it holds:*

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

We stress that, there exist different formulations of the UC framework, capturing different requirements on the set-assumptions (e.g., [10, 7]). In some formulation for example, the set-up assumption is global, which means that the environment has direct access to the set-up functionality $\mathcal{G}$ and therefore the simulator Sim needs to have oracle access to $\mathcal{G}$ as well. In [7] instead, while they assume that Sim cannot simulate (*program*) a PUF, and thus has always access to the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$, they require that $\mathcal{Z}$ has not permanent access to $\mathcal{F}_{\mathsf{PUF}}$.

Commitment Ideal Functionality $\mathcal{F}_{\mathsf{com}}$. The ideal functionality for Commitment Scheme is depicted in Fig. 5.

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

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

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

Figure 5: The Commitment Functionality $\mathcal{F}_{\mathsf{com}}$.

A.3 $\mathcal{F}_{\mathsf{PUF}}$ Ideal Functionality for Malicious PUFs

A malicious PUF is any physical device that "looks like" a PUF but it does not satisfy the PUF's security property. Namely, a malicious PUF could implement any function chosen by the adversary, and it can be stateful. The ideal functionality modeling malicious PUFs has been proposed in [37], and is the direct extension of the ideal functionality introduced in [7]. The PUF access model assumed by [37], follows the same model proposed in [7] and consists in the following. The simulator Sim has interface access to $\mathcal{F}_{\mathsf{PUF}}$. This means that Sim cannot simulate a PUF, but it has permanent oracle access to the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$. The environment has a restricted access to $\mathcal{F}_{\mathsf{PUF}}$ in the following sense. It can invoke command Eval of $\mathcal{F}_{\mathsf{PUF}}$ (i.e., query the PUF) only in case the PUF is in possession of the dummy adversary, or when the PUF is in transit. Additionally, the dummy adversary and thus also the simulator, have the power of creating honest PUFs.

The ideal functionality of [37] is depicted in Fig. 6. $\mathcal{F}_{\mathsf{PUF}}$ is parametrized by one honest PUF family and one malicious PUF family. In our construction we need two PUFs that have different parameters. This is not a problem, since $\mathcal{F}_{\mathsf{PUF}}$ can be straightforwardly extended so that it is parametrized by more then one honest PUF family. For more details about the model the reader is referred to [37].

A.4 $\mathcal{F}_{\mathsf{wrap}}$ Ideal Functionality modeling Stateless Tokens

The original work of Katz [27] introduces the ideal functionality $\mathcal{F}_{\mathsf{wrap}}$ to model stateful tokens in the UC-framework. A stateful token is modeled as a Turing machine. In the ideal world, a party that wants to create a token, sends the Turing machine to $\mathcal{F}_{\mathsf{wrap}}$. The adversary is, of course, allowed to send an arbitrarily malicious Turing machine to $\mathcal{F}_{\mathsf{wrap}}$. This translates in the fact that the adversary can send a malicious token to the honest party. $\mathcal{F}_{\mathsf{wrap}}$ will then run the machine (keeping the state), when the designed party will ask for it. The same functionality can be adapted

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

- When a party $\hat{P} \in \mathbb{P} \cup \{ \mathcal{S} \}$ writes $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, \mathtt{mode}, \hat{P})$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$, where $\mathtt{mode} \in \{ \mathtt{normal}, \mathtt{mal} \}$, then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathsf{sid}, *, *, *, *)$:
  - If this is the case, 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 $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, P_i, s)$ 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 $\sigma_S \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, s)$. Write $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, s, \sigma_S)$ 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 $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S}, s)$ 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 $\sigma_S \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, s)$ and return $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, s, \sigma_S)$ 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 6: The ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ for malicious PUFs.

to model stateless tokens. It is sufficient that the functionality does not keep the state between two executions.

One technicality of the model proposed by [27] is that it assumes that the adversary knows the code of the tokens that she sends. In real life, this translates to the fact that an adversary cannot forward tokens received from other parties, or tamper with its own token, so that the actual behavior of the token is not known to anyone. The advantage of this assumption, is that in the security proof the simulator can *rewind* the token.

In [13], Chandran, Goyal and Sahai, modify the original model of Katz, so to allow the adversary to create tokens without knowing the code. Formally, this consists in changing the 'create' command of the $\mathcal{F}_{\mathsf{wrap}}$ functionality, which now takes as input an Oracle machine instead of a Turing machine.

The model of [13] is even stronger and allows the adversary to encapsulate tokens.

Our security proofs are unconditional, and our simulator and extractor only exploit the interface access to the ideal functionality $\mathcal{F}_{\mathsf{wrap}}$ (i.e., they only observe the queries made by the adversary), namely, they do not need adversary's knowledge of the code. Therefore, our proofs hold in both [13] and [27] models. In this work, similarly to all previous work on stateless tokens [30, 25, 15], and also [24], we do not consider adversaries that can perform token encapsulation. To sum up, a malicious token is a physical device that "looks like" a token but implements a functionality which is arbitrarily different from the one dictated by the protocol. It is assumed that once a malicious token is sent away to the honest party, it cannot communicate with its creator.

A simplification of the $\mathcal{F}_{\mathsf{wrap}}$ functionality as shown in [13] (that is very similar to the $\mathcal{F}_{\mathsf{wrap}}$ of [27]) is depicted in Fig. 7.

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

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

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

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

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

Figure 7: The $\mathcal{F}_{\mathsf{wrap}}$ functionality.

B Properties of Protocol BobEquality

For convenience protocol BobEquality is rewritten below. Recall that a blob is a pair of commitments to bit, and the value committed to in a blob is the xor of such bits. Namely, a blob $\mathbf{B}_i$ is the pair $(\mathbf{c}_i^0, \mathbf{c}_i^1)$, of commitments of bits (b_i^0, b_i^1), and the values committed to in blob $\mathbf{B}_i$ is the bit $b_i = b_i^0 \oplus b_i^1$. For simplicity in the following we use b_i, b_j to denote "the bit committed in blob $\mathbf{B}_i, \mathbf{B}_j$".

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

1. V uniformly chooses $e \in \{0, 1\}$ and commits to e using IdealExtCom.
2. P sends $y = b_i^0 \oplus b_j^0$ to V
3. V reveals e to P.
4. P reveals b_i^e and b_j^e. V accepts iff $y = b_i^e \oplus b_j^e$.

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

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

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

Lemma 2 (Privacy of BobEquality)**.** *Assume that $\mathbf{B}_i, \mathbf{B}_j$ are statistically hiding commitments. If $b_i = b_j$ then for any malicious verifier V^* the view is independent on the actual value of b_i and b_j.*

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

Lemma 3 (Simulation of BobEquality in the $\mathcal{F}_{\mathsf{aux}}$ model)**.** *If* IdealExtCom *is a straight-line extractable commitment in the $\mathcal{F}_{\mathsf{aux}}$-hybrid model, then there exists a straight-line PPT algorithm* SimFalse*, called simulator, such that for any V^*, the view of V^* interacting with* SimFalse *on input a pair $(\mathbf{B}_i, \mathbf{B}_j)$ of possibly not equal blobs (i.e., $b_i \neq b_j$) is statistically close to the view of V^* when interacting with P and $b_i = b_j$.*

Proof. In the following we use the assumption that blobs are statistically hiding, therefore given $\mathbf{B}_i, \mathbf{B}_j$, any pair b_i, b_j is equally likely to be the committed values. Let E be the straight-line extractor associated to IdealExtCom as required by Definition 3. On common input $(\mathbf{B}_i, \mathbf{B}_j)$, SimFalse has interface access to $\mathcal{F}_{\mathsf{aux}}$ and works as follows.

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

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

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

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

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

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

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

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

Lemma 4 (Indistinguishability of the Simulation after one blob is revealed.)**.** *The view of V^* in the simulated execution (where $b_i \neq b_j$) is statistically close to the view of V^* in the honest execution (where $b_i = b_j$) even if, at the end of the protocol, one blob is revealed.*

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

are randomly chosen). Given the equation:

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

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

□

C UC-security of UCComCompiler

In this section we provide formal proof of Theorem 1. We show a straight-line simulator Sim having interface access to $\mathcal{F}_{\mathsf{aux}}$ and interacting with $\mathcal{F}_{\mathsf{com}}$ only, that for any environment $\mathcal{Z}$, generates a transcript that is indistinguishable from the transcript that $\mathcal{Z}$ obtains from the real-world adversary $\mathcal{A}$ participating (or just observing) the real protocol execution. We distinguish three cases, according to which party $\mathcal{Z}$ corrupts, if any.

C.1 Committer and Receiver are honest

In this case the real-world adversary $\mathcal{A}$ is instructed by $\mathcal{Z}$ to not corrupt any party. The goal of the simulator is to generate the transcript of the interaction between honest parties $\mathsf{C_{uc}}, \mathsf{R_{uc}}$. The procedure of Sim is described in Simulator 1.

Simulator 1. *[Sim in the honest-honest case.]*

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

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

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

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

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

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

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

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

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

Claim 1 (Indistinguishability of the simulation when both parties are honest)**.** *If blobs are ideal commitments, for any real-world adversary* $\mathcal{A}$ *and any environment* $\mathcal{Z}$*, the transcript generated by* Sim *(Simulator 1) is statistically indistinguishable from the interaction between honest real-world* $\mathsf{C_{uc}}, \mathsf{R_{uc}}$.

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

In the honest-honest case, the environment $\mathcal{Z}$ sets the input of the honest sender $\mathsf{C_{uc}}$, observes the communication between $\mathsf{C_{uc}}$ and $\mathsf{R_{uc}}$, and possibly delays/drops messages (we assume authenticated channel) of the protocol through the dummy adversary $\mathcal{A}$. We show that the transcript simulated by Sim 1 is statistically close to the actual transcript obtained from the real interaction of honest $\mathsf{C_{uc}}, \mathsf{R_{uc}}$. The proof goes by hybrids arguments. It starts from the real world, hybrid H_0, in which $(\mathsf{C_{uc}}, \mathsf{R_{uc}})$ honestly run the protocol using the input received from $\mathcal{Z}$, and it ends to the ideal world, hybrid H_4, where Sim simulates both parties without knowing the actual input.

Hybrid H_0: This is the real world.

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

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

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

In hybrid H_2^{j-1} the j-th execution of BobEquality is executed following the procedure of the honest prover P. In hybrid H_2^j, the procedure of a modified (the challenge e do not need to be extracted) SimFalse is followed instead. By lemma 2, it holds that the transcript generated by SimFalse is statistically close to the transcript generated by an honest prover. In our case is even identical since we do not have to consider the negligible probability of failure of the extraction, and since the pair of blob $\mathbf{B}_j, \mathbf{B}_{j-1}$ are equal.

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

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

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

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

The difference between hybrids H_3 and H_4 is that blobs do not commit to the same bit, they are not all equal. Therefore, in H_4 the simulated proofs are given on pairs of blobs that are not equal, and then one of the blobs is revealed. By Lemma 4, and the statistically hiding property of blobs (that are ideal commitment schemes) it follows that hybrids H_3 and H_4 are statistically close.

Noticing that Sim_4 corresponds to the procedure of Sim (Simulator 1), we have that hybrid H_4 is the ideal world. The claim is proved.

□

C.2 Receiver is corrupt

In this case the environment $\mathcal{Z}$ instructs the real-world adversary $\mathcal{A}$ to corrupt the receiver R_{uc}. The simulator in this case, is very close to Simulator 1 shown for the honest-honest case. Therefore we will just point out the differences with the previous simulator, and how the same indistinguishability proof can be consequently adapted.

Concerning the simulator, the main difference with Simulator 1 is in Step 2. While in the honest-honest case the challenge is chosen by Sim 1 itself, in the malicious receiver case, the challenge must be extracted from the adversary. This simply means that Step 2 must be replaced with procedure SimFalse shown in Lemma 2. Furthermore, the simulator in this case is not simulating R_{uc}, but is internally running $\mathcal{A}$ that plays as a receiver. Thus, it has to take care of $\mathcal{A}$ aborting the protocol at any point.

The proof that such simulation is indistinguishable from the real-world execution goes along the same lines of the proof provided for the honest-honest case. The main difference is in hybrid H_2, that in case of malicious receiver, is only statistically close to hybrid H_1. Indeed, when the receiver is malicious we have to consider the negligible probability of the failure of the extractor associated to the commitment scheme IdealExtCom.

C.3 Committer is corrupt

In this case, the environment $\mathcal{Z}$ instructs the adversary $\mathcal{A}$ to corrupt the sender C_{uc}. The simulator Sim internally simulates a copy of the dummy adversary $\mathcal{A}$ as well as $\mathcal{A}$'s communication with $\mathcal{Z}$.

In addition, Sim simulates the honest receiver $\mathsf{R_{uc}}$ to $\mathcal{A}$. The goal of Sim is to extract the bit that $\mathcal{A}$ is committing to in the simulated execution, so that it can send it to the ideal functionality $\mathcal{F}_{\mathsf{com}}$.

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

Simulator 2 (Sim in case sender $\mathsf{C_{uc}}$ is corrupt.)**.** Activate $\mathcal{A}$ on input the security parameter n and the secret bit received by $\mathcal{Z}$. When $\mathcal{A}$ starts the commitment phase, proceeds as follows.

Commitment Phase.

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

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

3. *If all proofs are successful, consider the bits extracted in Step 1, and check which case applies:*
   1. *There exists a bit b such all adjacent pairs of extracted bit are either (b, b) or $(b, \bar{b})$. In this case, since in the decommitment phase $\mathcal{A}$ is required to open one bit for each pair, there is only one bit that $\mathcal{A}$ can possibly decommit to, and is the bit b. Thus, send* (commit, sid, $\mathsf{C_{uc}}$, $\mathsf{R_{uc}}$, b) *to $\mathcal{F}_{\mathsf{com}}$.*
   2. *There exists at least an adjacent pair of bits (b, b) and at least one pair of bits $(\bar{b}, \bar{b})$. In this case, $\mathcal{A}$ that has to open at least one bit for each pair, cannot successfully commit to any bit. Thus send* (commit, sid, $\mathsf{C_{uc}}$, $\mathsf{R_{uc}}$, 0) *to $\mathcal{F}_{\mathsf{com}}$.*
   3. *(***Failure***) Each adjacent pair is either $(0, 1)$ or $(1, 0)$. In this case, $\mathcal{A}$ could correctly decommit to both* 0 *and* 1*. Thus, abort. We call this event* Input Extraction Failure.

Decommitment phase.

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

Claim 2 (Indistinguishability of the simulation when the sender is corrupt). *If blobs are ideal* extractable *commitments, for any real-world adversary $\mathcal{A}$ corrupting the sender $\mathsf{C_{uc}}$, any environment $\mathcal{Z}$, it holds that view $\mathsf{REAL}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{UCCom},\mathcal{A},\mathcal{Z}}$ is statistically close to $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}\ 2,\mathcal{Z}}$.*

Proof. Sim 2 behaves almost identically to honest receiver $\mathsf{R_{uc}}$. Indeed, it runs E in the first step, that due to the simulation property of IdealExtCom, generates a view that is identical to the one generated by an honest receiver. Then it honestly follows protocol BobEquality. However, differently from the honest receiver, Sim 2 aborts more often. Specifically, Sim 2 additionally aborts in the following two cases:

Case 1. In Step 1, when the extractor E fails in extracting the bit from any of the blobs.

Case 2. In Step 3, Sim fails in determining the bit committed to by $\mathcal{A}$. We call this event *Input extraction Failure*, since Sim fails in extracting the input to send to the ideal functionality $\mathcal{F}_{\mathsf{com}}$.

Case 3. In the decommitment phase $\mathcal{A}$ opens to a bit b that is different from the one extracted by Sim.

Due to the extractability property of the ideal extractable commitment IdealExtCom, Case 1 happens only with negligible probability. Due to Lemma 5, probability of Case 2 is also negligible. Finally, due to the statistically binding property of Blobs, probability that $\mathcal{A}$ can open to a bit that is different from the one extracted is negligible. Therefore, the view of $\mathcal{A}$ simulated by Sim is statistically close to the view obtained from the interaction with real world receiver. Which implies that the distribution of the input extracted by Sim is statistically close to the distribution of the input played in the real world, and the communication between $\mathcal{A}$ and $\mathcal{Z}$ simulated by Sim is also statistically close to the communication of $\mathcal{Z}$ with $\mathcal{A}$ interacting in the real protocol. Which implies that $\mathsf{REAL}^{\mathcal{F}_{\mathsf{aux}}}_{\mathsf{UCCom},\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}\ 2,\mathcal{Z}}$ are statistically close. □

Lemma 5. *Probability of event* Input extraction Failure *is negligible.*

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

Event 1: all executions of protocol BobEquality are successful. Namely, for all i [5], BobEquality($\mathbf{B}_i$, $\mathbf{B}_{i+1}$) provided by $\mathcal{A}$ is accepting.

Event 2: Each consecutive pair of blobs is not equal. Namely, for all i, $b_i \neq b_j$, where b_i and b_j are the bits committed respectively in $\mathbf{B}_i$, $\mathbf{B}_{i+1}$.

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

D Ideal Commitment Scheme of [37]

In Fig. 8 is depicted the ideal extractable commitment scheme based on (malicious) PUFs and presented in [37]. $\mathcal{P}_S$ denote the sid for accessing the PUF created by $\mathcal{F}_{\mathsf{PUF}}$ functionality.

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

Protocol CPuf

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

1. Committer $\mathsf{C}_{\mathsf{CPuf}}$: send $(\mathsf{init}_{\mathsf{PUF}}, \mathtt{normal}, \mathcal{P}_S, \mathsf{C}_{\mathsf{CPuf}})$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain response $(\mathsf{initialized}_{\mathsf{PUF}}, \mathcal{P}_S)$. Select a random string $s \in \{0,1\}^n$, send $(\mathsf{eval}_{\mathsf{PUF}}, \mathcal{P}_S, \mathsf{C}_{\mathsf{CPuf}}, s)$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtain response $(\mathsf{response}_{\mathsf{PUF}}, \mathcal{P}_S, s, \sigma_S)$. Compute $(st_S, p_S) \leftarrow \mathsf{FuzGen}(\sigma_S)$, and send p_S to $\mathsf{R}_{\mathsf{CPuf}}$ and $(\mathsf{handover}_{\mathsf{PUF}}, \mathcal{P}_S, \mathsf{C}_{\mathsf{CPuf}}, \mathsf{R}_{\mathsf{CPuf}})$ to $\mathcal{F}_{\mathsf{PUF}}$.

2. Receiver $\mathsf{R}_{\mathsf{CPuf}}$: obtain p'_S from the committer and $(\mathsf{handover}_{\mathsf{PUF}}, \mathcal{P}_S, \mathsf{C}_{\mathsf{CPuf}})$ from $\mathcal{F}_{\mathsf{PUF}}$. Pick random string $r \in \{0,1\}^\ell$ and send it to the committer.

3. Committer $\mathsf{C}_{\mathsf{CPuf}}$: send $\mathbf{c} = st_S \oplus (r \wedge b^\ell)$ to $\mathsf{R}_{\mathsf{CPuf}}$.

Decommitment Phase

1. Committer $\mathsf{C}_{\mathsf{CPuf}}$: send (b, s) to $\mathsf{R}_{\mathsf{CPuf}}$.

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

Figure 8: CPuf: Ideal Commitments in the $\mathcal{F}_{\mathsf{PUF}}$ model [37].

From Bit Commitment to String Commitment. To commit to a k-bit string one executes protocol CPuf k times in parallel and the receiver accepts the string iff all executions are accepting. The PUF family required for a k-bit string commitment is with challenge space n and range $k3n$ (one can construct such PUF by combining several PUFs together and querying them with the same challenge and concatenating the responses). Indeed, the binding argument relies on the expansion factor between PUF-challenge and PUF-response. In case of k-composition the expansion factor must be $k3n$ instead of $3n$, since for each $i \in [k]$, x_i might convey n bits of information to a malicious stateful PUF. Hiding holds since probability that a malicious receiver guesses one of the challenges is $k/2^n$.

E Ideal Extractable Commitments: Proofs

In this section we provide formal proofs of our ideal extractable commitments shown in Section 4 and Section 5.

E.1 Proof of the Ideal Extractable Commitment with PUFs: Protocol ExtPuf

In this section we formally prove Theorem 2. Namely, we prove that protocol ExtPuf (shown in Fig. 2) is an ideal extractable commitment scheme.

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

Hiding. The commitment phase of protocol ExtPuf basically consists in the parallel execution of two instances of CPuf. In the first instance, that we call ComBit, C_{ExtPuf} commits to its secret bit b, in the other instance, that we call ComResp, it commits to some value received from the (possibly malicious) PUF $\mathcal{P}_R^*$ [6]. Although $\mathcal{P}_R^*$ could compute the response *adaptively* on the query observed, thus revealing information about the opening (recall that the query corresponds to the opening of ComBit), such information cannot reach $\mathcal{A}$ since the response is committed using CPuf. Furthermore in case $\mathcal{P}_R^*$ aborts, C_{ExtPuf} continues the protocol, committing to the string 0, in fact, ruling out selective abort attacks.

Formally, the hiding proof goes by hybrids:

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

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

H_2 : In this experiment the commitment runs ComBit as commitment of 1 and ComResp still as commitment of 0^ℓ. Due to the hiding of CPuf, H_1 and H_2 are statistically close.

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

Binding. Binding follows straight-forwardly from the binding property of CPuf.

Extractability. We show a straight-line PPT extractor E that having interface access to $\mathcal{F}_{\mathsf{PUF}}$ satisfies the properties of Definition 3. The extractor is formally described in Fig. 9. $\mathcal{A}$ denotes the malicious sender.

Extractor E satisfies the following properties.

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

Simulation. The extractor E follows the procedure of the honest receiver R_{ExtPuf}, and additionally it collects the queries made by $\mathcal{A}$ to $\mathcal{P}_R$. Therefore the view of $\mathcal{A}$ interacting with E is identical to the view of $\mathcal{A}$ interacting with R_{ExtPuf}.

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

Extractor E

E creates PUF $\mathcal{P}_R$ sending $(\mathsf{init}_{\mathsf{PUF}}, \mathtt{normal}, \mathcal{P}_R, \mathsf{R}_{\mathsf{ExtPuf}})$ to $\mathcal{F}_{\mathsf{PUF}}$. E handovers the PUF to $\mathcal{A}$, sending $(\mathsf{handover}_{\mathsf{PUF}}, \mathcal{P}_R, \mathsf{R}_{\mathsf{ExtPuf}}, \mathcal{A})$ to $\mathcal{F}_{\mathsf{PUF}}$. Queries made by $\mathcal{A}$ to $\mathcal{P}_R$ are intercepted by E, stored in the variable $\mathcal{Q}$, and then forwarded to $\mathcal{F}_{\mathsf{PUF}}$. The answers received by $\mathcal{F}_{\mathsf{PUF}}$ are then forwarded to $\mathcal{A}$.

Commitment Phase:

E honestly follows the procedure of $\mathsf{R}_{\mathsf{ExtPuf}}$. If the commitment phase is accepting, E proceeds to extraction phase. Else, it halts. Let $(r, \mathbf{c})$ be the transcript of ComBit.

Extraction Phase:

- If there exists a query $x \in \mathcal{Q}$ such that $\mathbf{c} = \mathsf{Decode}(x)$ then output 0.
- If there exists a query $x \in \mathcal{Q}$ such that $\mathbf{c} = \mathsf{Decode}(x) \oplus r$ then ouput 1.
- Case 1) If there exist queries $x_0, x_1 \in \mathcal{Q}$ s.t. $\mathbf{c} = \mathsf{Decode}(x_0)$ AND $\mathbf{c} = \mathsf{Decode}(x_1) \oplus r$ then output $\perp$.
- Case 2) If there exist no query in $\mathcal{Q}$ that decodes to a valid opening of $\mathbf{c}$, output $\perp$.

Figure 9: E: Extractor associated to ExtPuf.

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

- Case 1. Note that, $\mathcal{A}$ can always compute x_0, x_1 such that $r = \mathsf{Decode}(x_0) \oplus \mathsf{Decode}(x_1)$ and compute $\mathbf{c} = \mathsf{Decode}(x_0)$. We argue that, if $\mathcal{A}$ computes $\mathbf{c}$ in such a way, then probability that $\mathcal{A}$ can provide an accepting decommitment for $\mathbf{c}$ is negligible. This follows directly from the binding of CPuf.
- Case 2. Towards a contradiction, assume that $\mathcal{A}$ does not query the PUF with any valid opening, but in the decommitment phase, $\mathcal{A}$ still provides an accepting decommitment. An accepting decommitment in ExtPuf consists of the decommitments of ComBit and ComResp. Namely, the bit b, along with the value st_S such that $\mathbf{c} = st_S \oplus (r \wedge b)$, and the string $(st_R || p_R)$ (for simplicity we are omitting the remaining decommitment data).

  Since the decommitment is accepting it holds that st_R is the answer of the **honest** PUF $\mathcal{P}_R$ on the query $\mathsf{Encode}(st_S)$(more precisely $st_R = \mathsf{FuzRep}(\sigma_R, p_R)$ where σ_R is the actual answer of $\mathcal{P}_R$ on input $\mathsf{Encode}(st_S)$).

  By hypothesis no queries received by $\mathcal{P}_R$ in the commitment phase decoded to st_S. Thus one of these two cases has happened:

  1. $\mathcal{A}$ has correctly computed $\mathcal{P}_R$'s responds σ_R without querying $\mathcal{P}_R$. In this case $\mathcal{A}$ breaks unpredictability of the honest PUF $\mathcal{P}_R$.

     Indeed, due to the Minimum Distance property of ECC, we have that all the valid codewords are at d_{min} hamming distance from each other. Thus, the only way for $\mathcal{A}$ to obtain a response for an encoding of st_S that was not inferred by E, is that such encoding is d_{min} apart from any challenge observed by E. Predicting the PUF-response of a challenge that is so far from the previously queried challenges, corresponds to break the unpredictability of the PUF.

2. $\mathcal{A}$ queries $\mathcal{P}_R$ only in the decommitment phase. Then she opens the commitment of the response, $\mathsf{ComResp}$, accordingly. Due to the statistically binding property of CPuf, this case happens with negligible probability.

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

□

E.2 Proof of the Ideal Extractable Commitments with Stateless Tokens: Protocol ExtTok

In this section we provide a formal proof of Theorem 3.

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

Theorem 4. *Protocol* CTok *is an ideal commitment scheme in the* $\mathcal{F}_{\mathsf{wrap}}$ *model.*

Proof. **Completeness.** By inspection.

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

Binding. The proof of binding can be adapted from the proof of protocol $\mathsf{IdealCom}$ (due to [37]). It is sufficient to observe that a malicious PUF can be a malicious token.

□

We are now ready to prove Theorem 3.

Proof. **Completeness.** Due to the completeness of the one-time unconditional MAC and the completeness of the sub-protocol CTok.
Hiding. Follows directly from the hiding property of protocol CTok. The formal argument is similar to the one provided in the hiding proof of Section E.1, and is therefore omitted.
Binding. Follows directly from the binding property of protocol CTok.
Extractability. Extractability roughly follows from the binding of CTok and the unconditional one-time unforgeability of MAC. Details follow.

We show a straight-line PPT extractor E that having interface access to $\mathcal{F}_{\mathsf{wrap}}$ satisfies the properties of Definition 3. The extractor is formally described in Fig. 10. $\mathcal{A}$ denotes the malicious sender.

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

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

Extraction. We show that, probability that the extractor E outputs $\perp$ (i.e., it fails in extracting the bit) but the adversary $\mathcal{A}$ is instead able to provide an accepting decommitment is negligible. From Fig. 10 E fails in the extraction in two cases.

Extractor E

E simulates the creation of $\mathcal{T}_{\mathsf{R}}$. Queries made by $\mathcal{A}$ to $\mathcal{T}_{\mathsf{R}}$ are intercepted by E, stored in the variable $\mathcal{Q}$, and then answered faithfully (i.e., by following the code of an honest $\mathcal{T}_{\mathsf{R}}$).

Commitment Phase:

E honestly follows the procedure of $\mathsf{R}_{\mathsf{ExtTok}}$. If the commitment phase is accepting, E proceeds to the extraction phase. Else, it halts. Let $(r, \mathbf{c})$ be the transcript of ComBit.

Extraction Phase:

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

Figure 10: E: Extractor associated to ExtTok.

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

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

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

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

□