Basing Obfuscation on Simple Tamper-Proof Hardware Assumptions

Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges

Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

Code obfuscation is one of the most powerful concepts in cryptography. It could yield functional encryption, digital rights management, and maybe even secure cloud computing. However, general code obfuscation has been proven impossible and the research then focused on obfuscating very specific functions, studying weaker security definitions for obfuscation, and using tamper-proof hardware tokens to achieve general code obfuscation. Following this last line this work presents the first scheme which bases general code obfuscation of multiple programs on one single stateless hardware token.

Our construction is proven secure in the UC-framework and proceeds in three steps:

1. We construct an obfuscation scheme based on fully homomorphic encryption (FHE) and a hybrid functionality conditional decrypt, which decrypts the result of a homomorphic computation given a proof that the computation was performed as intended. One difficulty of the first step are possible decryptions errors in the FHE. These decryption errors can occur whenever the randomness for the encryption is chosen maliciously by the receiver of the obfuscated code. Such decryption errors then could make a real obfuscated computation distinguishable from a black box use of the non-obfuscated program.

2. Given two common reference strings (CRS) we construct a UC-protocol realizing the functionality conditional decrypt with a stateless hardware token. As the token is stateless it is resettable by a dishonest receiver and the proofs given to the token must be resettably sound. One additional difficulty occurs when the issuer of the token can be corrupted. A malicious token can be stateful and it cannot be prevented that it aborts after a hardwired number of invocations. To prevent adaptive behavior of a malicious token the data of the receiver has to be hidden from the token and the proofs given to the token must even hide the size of the program and the length of the computation.

3. Last we construct a protocol constructing a CRS with a stateless hardware token. Care has to be taken here to not let the token learn anything about the resulting CRS which could not be simulated, because the very same token will later be used in a protocol based on the security of this CRS.

Keywords: Obfuscation, Stateless Tamper-Proof hardware, Universal Composability, Universal Arguments, Fully-Homomorphic-Encryption

1 Introduction

The study of program obfuscation receives a lot of attention, because, on the one hand it would have tremendous implications for practical applications like software-protection, digital rights management and cloud-computing. On the other hand, obfuscation allows to delegate arbitrary cryptographic tasks to untrusted parties. This includes the construction of public-key encryption schemes from private-key encryption schemes (where encryption is delegated) or the elimination of random oracles in cryptographic protocols. Since [BGI+01] there has been a lot of research on the topic of program obfuscation. [BGI+01] formalizes the intuitive requirements for obfuscation in the notion of virtual black-box (VBB) obfuscation. VBB-obfuscation guarantees that nothing more can be learned from the obfuscated code of a program than what can be learned by black-box access to the functionality that is implemented by the program. However, [BGI+01] rules out the possibility of virtual VBB-obfuscation for a wide range of functionalities. Subsequently, research on obfuscation has essentially developed in three directions. In the first direction, one tries to identify classes of functionalities that can be obfuscated in the strong VBB-sense. Function classes that were shown to be VBB-obfuscatable include point functions [Can97, Wee05, CD08], access control systems [LPS04] and, in a slightly relaxed sense, functionalities that operate on ciphertexts [HRSV07, Had10]. The second direction tries to find weaker meaningful notions of obfuscation than VBB-obfuscation. However, there exist strong impossibility results even for such relaxed notions of obfuscation [GK05, GR07]. Finally, the third direction considers the problem of constructing VBB-obfuscators under setup-assumptions. These setup-assumption include the random-oracle model [LPS04] and tamper-proof hardware assumptions [GKR08, GIS+10, DKMQ11].

Previous work in the third line faces certain limitations. The work of [LPS04] realizes point-function obfuscators but does not lead to general purpose obfuscation. The construction of [GKR08] can be seen as an approach to general purpose obfuscation with honest sender based on the concept of one-time-programs. However, this inherently limits the number of executions of the obfuscated program and requires stateful hardware. [GIS+10] provide a construction of a general purpose obfuscation-scheme based on stateless rather than stateful tamper-proof hardware. Unfortunately, the construction of [GIS+10] requires a hardware-invocation for each gate of the obfuscated circuit. Moreover, if the sender may deliver malicious stateful hardware, then the required number of hardware-tokens grows proportionally with the circuit-size. [DKMQ11] provides a construction of one-time-programs secure against malicious senders, which requires only a single hardware token. However, since this construction is based on one-time-programs, it has similar limitations as [GKR08]. Specifically it requires a stateful hardware-token and allows only a limited number of program-executions.

1.1 Our Contribution

We proceed the third line of research by presenting the first efficient universally composable [Can01], general purpose obfuscation-scheme based on a single stateless tamper-proof hardware device. The scheme allows the execution of different obfuscated programs an arbitrary number of times, while only requiring a single hardware token. We achieve this result in three steps. In step 1, we present such a scheme in a hybrid-model where the receiver has access to a *conditional decryption functionality*. This allows the sender of the obfuscation-scheme to deliver fully homomorphically encrypted [Gen09, vDGHV10, BV11b, BV11a, BGV11] programs to the receiver. Such encrypted programs

can be evaluated by the receiver on any input of her choice. The evaluation of the encrypted programs is performed in a deterministic input-oblivious model of computation, e.g. universal oblivious Turing machines. Since the receiver does not know the secret key of the encryption-scheme, she can only obtain the evaluation result by invoking the conditional decryption functionality. The receiver has to prove to the conditional decrypt functionality that the decryption request was honestly computed. The functionality then checks this proof, and if it is valid outputs the decrypted value. In contrary to preceding work [GIS$^+$10], the workload of the conditional decryption functionality is independent of the length of the program evaluation. In addition, the conditional decrypt functionality can be reused an arbitrary number of times, which is impossible for approaches based on one-time-programs [GKR08, DKMQ11]. This first scheme is perfectly secure against malicious senders and computationally secure against malicious receivers.

In step 2, we UC-implement the conditional decryption functionality with stateless tamper-proof hardware and a common reference string (CRS). On a high level, the protocol proceeds as follows. The sender first commits himself to the inputs of the conditional decryption functionality and sends an allegedly stateless token. The conditional decryption functionality is then realized by an efficient 2-party computation between the receiver and the token. This 2-party computation is implemented as follows. The receiver first generates a pair of public and secret keys for a fully homomorphic encryption scheme. She then encrypts her inputs to the conditional decryption functionality and sends them together with a proof of knowledge of plaintext to the token. The token then evaluates the conditional decryption functionality homomorphically using the receivers encrypted inputs and its hardwired sender-inputs. It then sends the result together with a proof of correctness to the receiver. The receiver checks the proof and, if it is correct, decrypts his layer of encryption from the result. This construction is computationally secure against malicious sender and receiver respectively, when given a reusable CRS. The CRS is needed to implement the non-interactive proofs of knowledge used by both parties. Since we allow malicious stateful tokens, we cannot rule out the possibility that the token simply aborts after a certain number of invocations. Thus we need to model this defect in the ideal functionalities for obfuscation and conditional decryption. We model this effect by allowing the functionalities to have a counter of invocations.

Finally in step 3 we replace the trusted CRS by an efficient interactive offline initialization-phase between sender, receiver and token. For this the sender hardwires some random string x and an answer to a randomly chosen hard question into the token. Then the sender sends the hard question together with the token to the receiver, who replies with a random string y to the sender. The sender computes the CRS by $\mathsf{CRS} = x \oplus y$, signs it and sends it together with the answer to the hard question to the receiver. The receiver then proves to the token that she knows the answer to the hard question by a zero-knowledge argument and expects to receive x as reply from the token. If this x matches with the CRS sent by the sender, the signed CRS will be used as CRS for our second protocol.

All together, using the combined protocol, we obtain a reusable obfuscation-scheme based on a single stateless hardware that prohibits adaptive behavior by a malicious token.

1.2 Our Techniques

Our constructions can be seen as a setting of mutual delegation, where the token delegates the evaluation of the program to the receiver and the receiver delegates the decryption to the token

respectively. Hence both have to prove to each other that they performed the requested task correctly. The workload of the token is kept low and independent of the program's evaluation-time by utilizing an adapted version of universal arguments [BG02]. Based on universal arguments, we construct an adaptively sound argument system to prove certain statements. For the emulation-protocol for conditional decryption, we need to force the parties to know their inputs, while not revealing them to the other party. Hence we use non-interactive zero-knowledge proofs of knowledge (NIZKPoKs) [BFM88, FLS90, SCO$^+$01, GOS06]. As argument-system in our third protocol we use resettably sound zero-knowledge arguments of knowledge [BGGL01], as the honest token is stateless and therefore resettable. Moreover, there is a subtle issue regarding the correctness of the fully homomorphic encryption-scheme we employ. Since any behavior of a malicious party must be simulatable, we cannot allow the occurrence of decryption-errors. Gentry's [Gen09] definition of fully homomorphic encryption does not rule out schemes, for which the coins used in the encryption and evaluation can be chosen such that non-detectable decryption errors depend on the plaintext. Thus, there is no obvious strategy for the simulator to decide whether to fake a decryption error or not in its simulated output (since they are not detectable).

The fully homomorphic encryption-scheme of [Gen09] has no decryption error. However, recent more efficient constructions of fully homomorphic encryption based on LWE [BV11b, BV11a, BGV11] are prone to decryption errors. Therefore, we need a mechanism to deal arbitrary fully homomorphic encryption schemes (with negligible decryption-error). Finally, we have to address a technical issue. Since we are working in the UC-framework [Can01], an environment might communicate covert messages to the token through the length of the receiver-inputs. We avoid this by requiring a fixed upper bound for the receiver-input lengths and padding the inputs accordingly. This issue cannot be avoided in the UC-model by using compressing commitments since compressing commitments are not online-extractable, and thus do not compose.

1.3 Further Related and Concurrent Work

The increased interest in cloud-computing applications led to a series of works on the delegation of computation, which we briefly review in the following. [GGP10] construct a secure verifiable delegation scheme by combining fully homomorphic encryption and garbled circuits. [CKV10] improve the delegator's efficiency by not requiring her to garble a circuit. Based on an idea of [AIK10], [CKV10] exploit the secrecy of fully homomorphic encryption to imply correctness by mixing hidden challenges among the requests. Recently, [BCCT11, SG11, DFH11] suggested to check the correctness of an encrypted delegated computation by the use of a 2-round, private coin argument-system which are based on extractability assumptions. Unfortunately, such an approach has to fail in our setting, since a malicious receiver can always rewind the stateless hardware token in order to learn its private coins. The setting considered in [GS09] is similar to ours, since they also consider how a party can achieve secure computation by interacting with a stateless hardware token. However their scheme is not intended to achieve obfuscation since the token has almost the same complexity as the user. In contrast to that, in our construction the receiver carries out the bulk of the workload while the computation of the token is not even related to computation-length.

Concurrently and independently, [BCG$^+$11] constructed an hardware-based obfuscation scheme similar to ours. However, their focus lies on honest but leaky hardware, whereas our focus is on minimizing the number of tokens used and allowing the token to be stateful and corrupted. Finally, in another independent and concurrent work, [CKS$^+$11] investigated how stateless tamper-proof

hardware tokens can serve as a minimal UC-setup assumption, also applying similar techniques to ours.

A preliminary version of this work was presented at the Dagstuhl Workshop on Public-Key Cryptography 2011.

2 Preliminaries

Let in the following k denote a security-parameter. We use the cryptographic standard notions of negligible functions, as well as computational/statistical/perfect indistinguishability. We denote the assignment-operator with $\leftarrow$, while we use $=$ to denote equality.

2.1 Framework

We state and prove our results in the Universal-Composability (UC) framework of [Can01]. In this framework security is defined by comparison of a *real model* and an *ideal model*. The protocol of interest Π is running in the latter, where an adversary $\mathcal{A}$ coordinates the behavior of all corrupted parties. We assume static corruption, i.e. the adversary $\mathcal{A}$ cannot adaptively change corruption during a protocol-run. In the ideal model, which is secure by definition, an ideal functionality $\mathcal{F}$ implements the desired protocol task and a simulator $\mathcal{S}$ tries to mimic the actions of $\mathcal{A}$. An environment $\mathcal{Z}$ is plugged either to the ideal or the real model and has to guess, which model it is actually plugged to. Denote the random variable representing the output of $\mathcal{Z}$ when interacting with the real model by $\mathsf{Real}_{\Pi}^{\mathcal{A}}(\mathcal{Z})$ and when interacting with the ideal model by $\mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z})$. Protocol Π is said to be UC-secure if for any environment $\mathcal{Z}$ the distributions $\mathsf{Real}_{\Pi}^{\mathcal{A}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}}^{\mathcal{S}}(\mathcal{Z})$ are (computationally, statistically or perfectly) indistinguishable.

2.2 Universal Input-Oblivious Models of Computation

In order to be able to evaluate machines homomorphically, the underlying model of computation has to be deterministic and input-oblivious. We call a model of computation input-oblivious, if its sequence of elementary operations only depends on the size of the input but not on the input itself. This rules out models of computation that require control-flow like conditional jumps. In general, such models can be regarded as implicit and compact representations of circuits. A non-trivial model of computation with this property are oblivious Turing machines. For such Turing machines the head-movement is a fixed function of time [1]. [PF79] showed that a two tape oblivious Turing machine can simulate any non-oblivious Turing machine with only logarithmic slowdown. Henceforth, when we use the term machine or program M, we use it in the sense of a deterministic oblivious machine or a circuit. For a program M, define spec_M to be the auxiliary specification a universal machine requires to simulate M. For a circuit M, it is sufficient to define spec_M to specify the number of input and output-wires and the number of gates. For an oblivious Turing machine M, we can define spec_M to be a runtime-bound of M.

[1] Oblivious RAMs seem to be inappropriate for this application as their head-movement is random rather than fixed.

2.3 Fully Homomorphic Encryption

We give a brief overview of a fully homomorphic encryption without going into detail. A fully homomorphic encryption scheme FHE consists of four algorithms.

- $\mathsf{KeyGen}(1^k)$: Generates a public/secret-key pair (pk, sk)
- $\mathsf{Enc}_{pk}(m)$: Takes as input a public key pk and a message m and
- $\mathsf{Dec}_{sk}(\hat{m})$: Decrypts the output of the evaluated circuit using the secret key sk. Complexity is equivalent to Encrypt_E.
- $\mathsf{Evaluate}_{pk}(M, \hat{x})$: Takes as input the public key pk, a program M and the encrypted values $\hat{x}$. Returns an encrypted representation of $M(x)$.

The algorithms KeyGen, Enc and $\mathsf{Evaluate}$ are probabilistic. When we want to express that one of them, for instance Enc uses specific coins r, we shall write $\mathsf{Enc}_{pk}(\cdot; r)$. When fixing r to be the all-zero coins of appropriate length, we write $\mathsf{Enc}_{pk}(\cdot; 0^*)$. We will usually denote ciphertexts with a hat-notation, i.e. $\hat{m} = \mathsf{Enc}_{pk}(m)$. For larger plaintexts $\mathbf{m} = (m_1, \ldots, m_n)$, where each m_i is in the domain of FHE, we write $\mathsf{Enc}_{pk}(\mathbf{m}) = (\mathsf{Enc}_{pk}(m_1), \ldots, \mathsf{Enc}_{pk}(m_n))$. Even though the $\mathsf{Evaluate}$ function is usually defined on circuits [Gen09], we note that other input-oblivious models of computation can be seen as compact representations of circuits. Thus we can extend the $\mathsf{Evaluate}$ function to such models of computation without any difficulty. Further, we will use the $\mathsf{Evaluate}$-function in a rather liberal we. Specifically, let $\tilde{M} = (\hat{M}, \mathsf{spec}_M)$, where $\hat{M}$ is the encryption of a machine M and spec_M is its specification (e.g. for circuits number of gates, for oblivious Turing machines a runtime bound). Then we will also write $\mathsf{Evaluate}_{pk}(\tilde{M}, x)$ for the execution of the encrypted program $\hat{M}$ on input x, where x is either a plain- or ciphertext. If x is a plaintext, $\mathsf{Evaluate}$ will first encrypt x. Moreover, if the program M takes several arguments, we shall write $\mathsf{Evaluate}_{pk}(\tilde{M}, x_1, \ldots, x_n)$, where we allow the x_i to be either plain- or ciphertexts. We require fully homomorphic encryption schemes FHE to have circuit-privacy. FHE is defined to be circuit private, if for any public key pk, any program M and any input x it holds that $\mathsf{Evaluate}_{pk}(M, x) \approx_s \mathsf{Enc}_{pk}(M(x))$ (where $\approx_s$ denotes statistical indistinguishability). Thus, ciphertexts do not reveal whether they are the result of a homomorphic computation or mere encryptions of the computation-result. A fully homomorphic encryption scheme FHE is said to be correct, if for all admissible M and all x it holds that $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(x)) = x$ and $\mathsf{Dec}_{sk}(\mathsf{Evaluate}_p k(M, x)) = M(x)$ with overwhelming probability over the coins of KeyGen, i.e.

$$\Pr_{(pk,sk)\leftarrow\mathsf{KeyGen}(1^k)}[\forall M \forall x : \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(x)) = x \text{ and } \mathsf{Dec}_{sk}(\mathsf{Evaluate}_{pk}(M, x)) = M(x)] \geq 1 - \mathsf{negl}(k).$$

The schemes of [vDGHV10, BV11b, BV11a, BGV11] do not suffice this strong correctness condition. However, in the next section we introduce a slightly weaker correctness condition for fully homomorphic encryption-schemes, which is sufficient for our application. We use the standard IND-CPA security-definition for fully homomorphic encryption schemes. Informally the IND-CPA experiment proceeds as follows: An adversary $\mathcal{B}$ gets a public pk and has oracle access to a corresponding decryption oracle. He passes a pair of challenge-plaintexts m_0, m_1 to the experiment, which decides by coin toss if it encrypts $\hat{m} = \mathsf{Enc}_{pk}(m_0)$ or $\hat{m} = \mathsf{Enc}_{pk}(m_1)$. $\mathcal{B}$ wins the experiment if it correctly decides whether $\hat{m}$ encrypts m_0 or m_1. We say that an encryption-scheme FHE is IND-CPA secure, if no PPT-adversary $\mathcal{B}$ wins the IND-CPA experiment for FHE with advantage non-negligible better than $1/2$.

2.4 Strongly Unforgeable Signatures

As our scheme requires the use of signatures, we shall briefly review the standard-notion of strongly unforgeable signature-schemes. A signature-scheme SIG consists of three PPT-algorithms KeyGen and Sign and Verify.

- $\mathsf{KeyGen}(1^k)$ generates a public verification-key vk and a private signature-key sgk
- $\mathsf{Sign}_{sgk}(m)$ takes a signature-key sgk and a message $m \in \{0,1\}^*$ and returns a signature σ
- $\mathsf{Verify}_{vk}(m,\sigma)$ takes as input a verification-key vk, a message $m \in \sigma^*$ and a signature σ and outputs 1 if σ is a valid signature for m and 0 otherwise.

In the EUF-CMA-experiment an adversary $\mathcal{A}$ is given a verification key vk and access to a signature-oracle. $\mathcal{A}$ wins the experiment if it manages to forge a valid signature σ for a message of its choice m, without having queried its signature-oracle with m. A signature-scheme SIG is called EUF-CMA-secure, if no PPT-adversary $\mathcal{A}$ wins the EUF-CMA-experiment better than with negligible probability. For the sake of simplicity, we require signature-schemes with deterministic verification procedure and succinct signature length (i.e. the length of σ does not depend on m). Standard hash-and-sign [NY89, Rom90] constructions suffice these requirements.

2.5 Universal Arguments, Resettably Sound Zero-Knowledge and NIZKPoKs

For the construction of our obfuscation-scheme, we use the machinery of universal arguments (UA) [BG02], non-interactive zero-knowledge proofs of knowledge (NIZKPoK) [BFM88, FLS90, SCO$^+$01, GOS06] and resettably sound zero knowledge (rsZK) [BGGL01]. We briefly define the three notions.

Definition 1. Let $\mathcal{S}_U = \{(M,x,t) : \exists w \text{ s.t. Machine } M \text{ accepts } (x,w) \text{ within } t \text{ steps}\}$ be the *universal language*. Let $\mathcal{R}_U$ be the witness-relation of $\mathcal{S}_U$. A universal argument system is a pair of interactive Turing machine (P,V), such that the following conditions hold.

- **Efficient Verification** It exists a fixed polynomial $p(\cdot)$, such that for every $(M,x,t) \in \mathcal{S}_U$, the total time spent by the probabilistic verifier V is at most $p(|(M,x,t)|)$.
- **Completeness via relatively efficient prover** For every $((M,x,t),w) \in \mathcal{R}_U$, it holds that $\Pr[\langle \mathsf{P}(w),\mathsf{V}\rangle(M,x,t)=1]=1$. Furthermore there exists a polynomial $p(\cdot)$, such that for every $((M,x,t),w) \in \mathcal{R}_U$ the time spent by $\mathsf{P}(w)$ on common input (M,x,t) is at most $p(|M|+t)$.
- **Computational Soundness** For every non-uniform PPT P^* and every $(M,x,t) \notin \mathcal{S}_U$ with $|(M,x,t)|=n$ it holds that $\Pr[\langle \mathsf{P}^*,\mathsf{V}\rangle(M,x,t)=1] < \mathsf{negl}(n)$.
- **Weak Proof of Knowledge** For every polynomial $p(\cdot)$ there exists a polynomial $p\prime(\cdot)$ and a PPT oracle machine Ext, such that for every PPT P^* and every sufficiently long $y=(M,x,t) \in \mathcal{S}_U$ it holds that: If $\Pr[\langle \mathsf{P}^*,\mathsf{V}\rangle(M,x,t)=1] < 1/p(|y|)$ then $\Pr_r[\exists w \in \mathcal{R}_U(y) : \forall i \in [|w|] : \mathsf{Ext}^{\mathsf{P}^*}(y,i;r)=w_i] > 1/p'(|y|)$.

Universal argument systems can be instantiated for arbitrary $\mathcal{NTIME}(f(k))$-languages L, by setting M to be the witness-verifier of L and $t=O(f(k))$. The completeness condition says that the prover P has instance-based complexity, i.e. the runtime of the prover depends on the size of

the witness and can be super-polynomial if the witness has super-polynomial size. The universal argument system of [BG02] has the additional feature that it is public coin, i.e. the each message of the verifier consists *w.l.o.g* of a string of uniformly chosen coins.

Definition 2. A non-interactive zero-knowledge proof of knowledge system for a language $L \in \mathcal{NP}$ (with witness-relation $\mathcal{R}_L$) consists of a triple $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$ of PPT machines such that there exist three PPT-machines $\mathsf{FakeGen}$, Sim and Ext and the following conditions hold.

- $\mathsf{CRSGen}(1^k)$ samples a string CRS according to some distribution.
- $\mathsf{FakeGen}(1^k)$ samples a pair $(\tilde{\mathsf{CRS}}, td)$, such that the common reference string $\tilde{CRS}$ is statistically close to $\mathsf{CRS} \leftarrow \mathsf{CRSGen}(1^k)$.
- **Completeness** For every $(x, w) \in \mathcal{R}_L$ it holds $\Pr[\mathsf{V}(\mathsf{CRS}, x, \pi) = 1 | \mathsf{CRS} \leftarrow \mathsf{CRSGen}(1^k), \pi \leftarrow \mathsf{P}(\mathsf{CRS}, x, w)] = 1$.
- **Soundness** For every $x \notin L$ and every PPT machine P^* it holds that $\Pr[\mathsf{V}(x, \pi) = 1 | \mathsf{CRS} \leftarrow \mathsf{CRSGen}(1^k), \pi \leftarrow \mathsf{P}^*(\mathsf{CRS}, x)] < \mathsf{negl}(k)$.
- **Unbounded Zero Knowledge** For every PPT-machine D it holds that D cannot distinguish non-negligibly better than $1/2$ between the two experiments Real and Ideal. In the Real experiment a common reference string is generated by $\mathsf{CRS} \leftarrow \mathsf{CRSGen}(1^k)$. D is provided with CRS and unbounded oracle access to a prover $\mathsf{P}(\mathsf{CRS}, \cdot, \cdot)$, that takes as input $(x, y) \in \mathcal{R}_L$ and outputs $\mathsf{P}(\mathsf{CRS}, x, y)$. In the Ideal experiment a common reference string is generated by $(\mathsf{CRS}, td) \leftarrow \mathsf{FakeGen}(1^k)$. D is provided with CRS and oracle access to a simulator $\mathsf{Sim}(\mathsf{CRS}, td, \vdots, \cdot)$, which takes as input $(x, y) \in \mathcal{R}_L$ and outputs $\mathsf{Sim}(\mathsf{CRS}, td, x)$.
- **Proof of Knowledge** $\mathsf{Ext}(\tilde{CRS}, td, x, \pi)$ outputs a witness w s.t for every PPT-machine P^* it holds that $\Pr[\mathsf{V}(\mathsf{CRS}, x, \pi) = 1 \text{ and } (x, w) \notin \mathcal{R}_L | (\mathsf{CRS}, td) \leftarrow \mathsf{FakeGen}(1^k), (x, \pi) \leftarrow \mathsf{P}^*(\mathsf{CRS}), w \leftarrow \mathsf{Ext}(\mathsf{CRS}, td, x, \pi)] \leq \mathsf{negl}(k)$

Our construction will use NIZKPoKs with reusable uniformly chosen common reference strings [FLS90]. Note that any NIZK-system in the random CRS-model can be converted into a NIZKPoK-system in the random CRS-model. The transformation requires an IND-CPA secure public-key encryption scheme with dense public keys [RS92, NY90].

Definition 3. A resettably sound zero-knowledge argument of knowledge system for a language $L \in \mathcal{NP}$ (with witness-relation $\mathcal{R}_L$) consists of a pair of PPT-machines (P, V), where the verifier V is stateless, such that there exist two PPT-machines Sim and Ext and the following conditions hold.

- **Completeness** For every $(x, w) \in \mathcal{R}_L$ it holds that $\Pr[\langle \mathsf{P}(w), \mathsf{V} \rangle(x) = 1] = 1$.
- **Computational Soundness** For every $x \notin L$ and every PPT machine P^* it holds that $\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] < \mathsf{negl}(|x|)$.
- **Zero-Knowledge** For every $(x, w) \in \mathcal{R}_L$ and every stateful or stateless PPT V^* it holds the distributions $\mathsf{Real} = \{\langle \mathsf{P}(w), \mathsf{V}^* \rangle(x)\}$ and $\mathsf{Ideal} = \{\mathsf{Sim}(x, \mathsf{V}^*)\}$ are computationally indistinguishable.

- **Proof of Knowledge** For every $x \in L$ and every PPT-machine P^* with $\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle = 1$ and $(x, w) \notin \mathcal{R}_L | w = \mathsf{Ext}(x, \mathsf{P}^*] < \mathsf{negl}(|x|)$.

As in [BGGL01] we use the convention that V is the resettable next-message function of the verifier that takes as input a transcript of the conversation and returns public coins for its next message. We can assume *w.l.o.g* that the last message τ a prover P sends to a resettable verifier V before it accepts encodes the entire conversation between P and V.

Finally, we need the notion of a adaptive soundness for argument-systems [DN07].

Definition 4. An argument-system $(\mathsf{CRSGen}, \mathsf{P}, \mathsf{V})$ for a language L is called adaptively-sound, if every PPT-machine P^* has only negligible probability of winning the following experiment AS.

Experiment AS

- The experiment sets $\mathsf{CRS} \leftarrow \mathsf{CRSGen}(1^k)$
- $(x, a) \leftarrow \mathsf{P}^*(\mathsf{CRS})$.
- Run $s = \langle \mathsf{P}^*(a), \mathsf{V}(\mathsf{CRS}, x) \rangle$.
- P^* wins if $s = 1$ and $x \notin L$.

3 Fully Homomorphic Encryption with Fixed Random Coin Consumption

We would like the amount of random coins used by Enc and $\mathsf{Evaluate}$ to be independent of the size of the plaintexts with encrypt and circuits we evaluate. The construction to achieve this is pretty simple, we just replace the random coins used by Enc and $\mathsf{Evaluate}$ with pseudorandom coins generated from a fixed length seed. However, this construction cannot fulfill the strong correctness notions provided in literature. We will therefore first provide a slightly relaxed notion of correctness that is met by our construction. To avoid non-uniform hardness-assumptions we will first remove the quantifiers in the definition of correctness in favor of a game-based definition.

Definition 5. Let FHE be a fully homomorphic encryption scheme. We say that FHE is *correct against PPT-adversaries*, if every PPT machine $\mathcal{A}$ fails to win the following experiment except with negligible probability.

Experiment COR

- The experiment generates $(pk, sk) \leftarrow \mathsf{FHE.KeyGen}(1^k)$.
- Let $(C, x) \leftarrow \mathcal{A}(pk, sk)$.
- Let r, r' be random coins. If $\mathsf{FHE.Dec}_{sk}(\mathsf{FHE.Evaluate}_{pk}(C, \mathsf{FHE.Enc}_{pk}(x; r); r')) \neq C(x)$ output 1, otherwise 0.

This correctness-definition is only required to hold against polynomial-sized circuits. We don't care if decryption-errors occur with higher probability when the scheme is evaluated on super-polynomial-sized circuits [2].

We will now give a general construction that transforms an arbitrary fully homomorphic encryption scheme into one where the Enc and $\mathsf{Evaluate}$ algorithms only consume a fixed amount of k random coins, independent of their respective inputs.

Construction 6. Let FHE be a fully homomorphic encryption scheme and $\mathsf{prg}(l, s)$ be a pseudo-random generator that expands a k-bit seed s to a length l string. The modified scheme FHE' uses the same KeyGen and Dec algorithms, but the following Enc and $\mathsf{Evaluate}$ algorithms.

- $\mathsf{Enc}_{pk}(\mathbf{m}; s)$: Let $\mathbf{m} = (m_1, \ldots, m_l)$. Compute $(r_1, \ldots, r_l) \leftarrow \mathsf{prg}(l \cdot k, s)$. For $i = 1, \ldots, l$ compute $\hat{\mathbf{c}} \leftarrow \mathsf{FHE.Enc}_{pk}(m_i; r_i)$. Output $\hat{\mathbf{c}}$
- $\mathsf{Evaluate}_{pk}(C, \hat{x}; s)$: Let l be the amount of random coins required by $\mathsf{FHE.Enc}_{pk}(C, \hat{x})$. Compute $r \leftarrow \mathsf{prg}(l, s)$ and $\hat{y} \leftarrow \mathsf{FHE.Evaluate}_{pk}(C, \hat{x}; r)$. Output $\hat{y}$.

We now show that this modification preserves correctness against PPT-adversaries.

Lemma 7. *Assume the fully homomorphic encryption scheme* FHE *is correct against PPT-adversaries. Given that* prg *is a secure pseudorandom generator, the scheme* FHE' *given in Construction 6 is also correct against PPT-adversaries.*

Proof. Assume for contradiction that FHE' is not correct against PPT-adversaries. Then there exists a PPT-machine $\mathcal{A}$ that wins the $\mathsf{COR}_{\mathsf{FHE}'}$-experiment with non-negligible probability ϵ. However, since FHE is correct against PPT-adversaries and the keys of FHE and FHE' are identically distributed, it holds that $\mathcal{A}$ cannot win the $\mathsf{COR}_{\mathsf{FHE}}$ with a probability higher than $\epsilon/2$. We will use $\mathcal{A}$ to construct a PPT-distinguisher $\mathcal{D}$ that distinguishes the distribution generated by prg from the uniform random distribution with advantage $\epsilon/2$. $\mathcal{D}$ proceeds as follows.

- Generate $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k)$
- Run $(C, x) \leftarrow \mathcal{A}(pk, sk)$.
- Let be the amount of random coins required by $\mathsf{FHE.Evaluate}_{pk}(C, \mathsf{FHE.Enc}_{pk}(x))$. Send l to the distinguishing experiment and let (r, r') be the corresponding output.
- Check if $C(x) \neq \mathsf{FHE.Dec}_{sk}(\mathsf{FHE.Evaluate}_{pk}(C, \mathsf{FHE}.Enc_{pk}(x; r); r'))$. If so, output 1, otherwise 0.

Now, if the sample (r, r') returned by the PRG-experiment is distributed uniformly random, then $\mathcal{D}$'s output is identically distributed to the output of the $\mathsf{COR}_{\mathsf{FHE}}(\mathcal{A})$-experiment. Thus $\Pr[\mathsf{PRG}^0(\mathcal{D}) = 1] = \Pr[\mathsf{COR}_{\mathsf{FHE}}(\mathcal{A}) = 1] < \epsilon/2$. On the other hand, if the sample (r, r') returned by the PRG-experiment is pseudorandom, then $\mathcal{D}$'s output is identically distributed to the output of the $\mathsf{COR}_{\mathsf{FHE}'}(\mathcal{A})$-experiment, thus $\Pr[\mathsf{PRG}^1(\mathcal{D}) = 1] = \Pr[\mathsf{COR}_{\mathsf{FHE}'}(\mathcal{A}) = 1] \geq \epsilon$. Thus we conclude

$$\mathsf{Adv}_{\mathsf{PRG}}(\mathcal{D}) = |\Pr[\mathsf{PRG}^0(\mathcal{D}) = 1] - \Pr[\mathsf{PRG}^1(\mathcal{D}) = 1]| \geq \epsilon/2.$$

□

[2] In fact, one can construct schemes that meet this correctness-definition, but fail for the general definition

4 Modeling Stateless Hardware and Program-Obfuscation in the UC-Framework

In this Section, we will introduce the ideal functionalities for stateless hardware, program-obfuscation and conditional decryption. We use the standard definitions for stateless hardware according to [GIS+10]. For simplicity, we state the functionality in the two-party case where only a sender-machine S and a receiver-machine R are present. This definition allows the sender S to wrap a stateless machine T in a hardware token, and send this token to the receiver R who can query it an arbitrary (polynomial) number of times. Additionally, we allow an adversarial sender to input a malicious machine T^* which is stateful instead of stateless. For the sake of readability, we omit session and message identifiers.

Functionality $\mathcal{F}_{\mathsf{wrap}}^{\mathsf{stateless}}$ (parametrized by a security parameter k and a polynomial $p(\cdot)$)

- Create Upon receiving $(\texttt{create}, \mathsf{T})$ from S, where T is a Turing machine, send `create` to R and store T. *Additional adversarial input*: An adversarial sender can additionally input a flag bad that indicates that machine T is stateful.
- Execute Upon receiving $(\texttt{run}, w)$ from R, check if a `create`-message has already been sent by S, if not output $\perp$. If the flag bad is not set, run $\mathsf{T}(w)$ for at most $p(k)$ steps, and let m be the output ($m = \perp$ if T does not halt in $p(k)$ steps). Output m to R. If flag bad is set, check if a previous state st has been stored (if not set $\mathsf{st} = 0$) and compute $(m, \mathsf{st}) = \mathsf{T}(w, \mathsf{st})$. Store the new state st and output m to R.

The messages between $\mathcal{F}_{\mathsf{wrap}}^{\mathsf{stateless}}$ and R are delivered immediately without scheduling by the adversary. We model the ideal program-obfuscation $\mathcal{F}_{\mathsf{Obf}}$ in the following way. The sender's inputs consist of pairs $(\texttt{id}, M)$, where `id` is a unique identifier and M a program. Once the sender inputs $(\texttt{id}, M)$ into $\mathcal{F}_{\mathsf{Obf}}$, the functionality stores $(\texttt{id}, M)$ and outputs `id` to the receiver. The receiver can query $\mathcal{F}_{\mathsf{Obf}}$ with inputs $(\texttt{id}, x)$ and receives and output $M(x)$, if a tuple $(\texttt{id}, M)$ was stored and x is a proper input for M, otherwise $\perp$. However, in order to allow a receiver simulator to mount a simulation, it receives as auxiliary output the specification spec_M of M (recall that spec_M contains information like the number of gates of a circuit or a runtime bound for an oblivious Turing machine) of M together with `id`. Since it cannot be avoided that a stateful hardware keeps a counter of the number of its invocations, our functionality $\mathcal{F}_{\mathsf{Obf}}$ has to reflect this by having a counter too.

Functionality $\mathcal{F}_{\mathsf{Obf}}$ (parametrized by a security parameter k, initialized with a counter $\mathsf{cntr} = 0$).

- Setup *Adversarial input by sender*: A maximum number of invocations n_{invoc}. By default set $n_{\mathsf{invoc}} = \infty$.
- Obfuscate Upon receiving a message $(\texttt{Obfuscate}, \texttt{id}, M)$ from S, where M is a program, send (`obfuscation`, `id`) to R and store (`id`, M). *Auxiliary Output to Receiver-Simulator:* spec_M.
- Query Upon receiving $(\texttt{Query}, \texttt{id}, x)$ from R, find the unique stored tuple (`id`, M). If no such tuple exists, output $\perp$. Otherwise, compute $M(x)$ and send it to R. Increase cntr by 1, shut down if $\mathsf{cntr} = n_{\mathsf{invoc}}$.

In order to prove our result modularly, we introduce an intermediate functionality we call *Conditional Decryption* (CONDEC). CONDEC can be seen as an abstraction of stateless hardware that forces the sender to behave honestly in our protocol for $\mathcal{F}_{\mathsf{Obf}}$. It basically takes decryption-queries together with an (interactive) argument of its validity and decrypts after verifying correctness. Let FHE be a fully homomorphic encryption scheme and SIG be an EUF-CMA secure signature scheme and $\{H_i\}$ be a family of collision-resistant hash-functions $\{0,1\}^* \to \{0,1\}^k$. Let L be a language that consists of tuples $(k, H, pk, \hat{y}, x, \ell, c, s)$ such that there exists a string $\tilde{M}$ with $|\tilde{M}| = \ell$ and $c = H(\tilde{M})$ and $\hat{y} = \mathsf{FHE.Evaluate}_{pk}(\tilde{M}, x; r)$, where the last expression has a computation-trace that has at most length $k^{\log(k)}$. Since $|\tilde{M}| \leq k^{\log(k)}$, we can assume that $\ell = |\tilde{M}|$ is represented by a binary string of length $\log^2(k)$. Clearly, it holds that $L \in \mathcal{NTIME}(k^{\log k})$. Therefore, we can prove membership in L using a constant round universal argument system of knowledge UA. Note that the runtime of the verifier of UA only depends on the size of the statement $(k, H, pk, \hat{y}, x, \ell, c, r)$. It is thus independent of the machine $\tilde{M}$ since the runtime of $\tilde{M}$ can be upper bounded by $k^{\log(k)}$. We have an additional requirements for UA. Since our goal is to use stateless hardware to implement $\mathcal{F}_{\mathsf{Obf}}$, we require UA to be resettably sound. This can be achieved straightforwardly by a transformation given in [BGGL01], where the random coins used by the verifier are replaced with pseudorandom coins generated from the message-history using a pseudorandom function.

Let (P, V) be an argument-system for the language L based on the universal argument system UA. Let $L' = \{(k, pk, \hat{y}, x, \tilde{M}, r) : \hat{y} = \mathsf{FHE.Evaluate}_{pk}(\tilde{M}, x; r)\}$. Given the soundness of (P, V) and the collision-resistance of $\{H_i\}$, the following argument-system $(\mathsf{P}', \mathsf{V}')$ for L' is complete and adaptively sound.

Construction 8. (parametrized by a security-parameter k)

- $\mathsf{CRSGen}(1^k)$: Set $H \leftarrow_R \{H_i\}$. Output H
- Prover P': Upon input $(H, (k, pk, \hat{y}, x, \tilde{M}, s))$, compute $c = H(\tilde{M})$, $\ell = |\tilde{M}|$ and simulate $\mathsf{P}((k, H, pk, \hat{y}, x, \ell, c), \tilde{M})$
- Verifier V': Upon input $(H, (k, pk, \hat{y}, x, \tilde{M}, s))$ simulate $\mathsf{V}(k, H, pk, \hat{y}, x, \ell, c, s)$

Completeness of Construction 8 follows from the completeness of the argument-system (P, V). We will prove adaptive soundness of Construction 8 in the next Subsection.

We want to avoid that the encrypted program $\tilde{M}$ has to be given to the CONDEC functionality, because this would induce too much overhead to the token. Moreover, the token would not be oblivious to the program actually executed. This would provide the environment with a covert channel into the token.

The hybrid-functionality $\mathcal{F}_{\mathsf{CONDEC}}$ used in our protocol Π_{Obf} utilizes the argument system (P, V). However, to establish the security-reduction for Π_{Obf} we will use the adaptive soundness of $(\mathsf{P}', \mathsf{V}')$. Finally, for the same reason as for $\mathcal{F}_{\mathsf{Obf}}$, we need to include a counter into $\mathcal{F}_{\mathsf{CONDEC}}$. Let pk be a public key of FHE and vk be a verification-key of SIG and H a hash-function from the family $\{H_i\}$ and F a pseudorandom function from a family of pseudorandom functions $\{F_i\}$. Let state_0 be private coins for the stateless verifier V.

Functionality $\mathcal{F}_{\mathsf{CONDEC}}$ (parametrized by a security parameter k, initialized with a counter $\mathsf{cntr} = 0$)

- Setup: Upon receiving a message $(\texttt{Setup}, (\mathsf{state}_0, F, H, vk, sk))$ from S, store $(\mathsf{state}_0, F, H, vk, sk)$ and send `ready` to R. *Additional adversarial input*: A maximum number of invocations n_{invoc}. By default set $n_{\mathsf{invoc}} = \infty$.
- Coins: Upon receiving a message $(\texttt{coins}, (x, c))$ from the receiver R, compute $s \leftarrow F(x, c)$ and return s to R. Increase cntr by 1, shut down if $\mathsf{cntr} = n_{\mathsf{invoc}}$.
- Verifier: Upon receiving a message $(\texttt{Verifier}, (\hat{y}, x, \ell, c, \tau))$ from the receiver R, compute $s \leftarrow F(x, c)$ and $r \leftarrow \mathsf{V}(\mathsf{state}_0, (k, H, pk, \hat{y}, x, \ell, c, s), \tau)$ and output r. Increase cntr by 1, shut down if $\mathsf{cntr} = n_{\mathsf{invoc}}$.
- Decrypt: Upon receiving a message $(\texttt{Decrypt}, (\hat{y}, x, \ell, c, \sigma, \tau))$ from the receiver R, compute $s \leftarrow F(x, c)$ and check if both $\mathsf{V}(\mathsf{state}_0, (k, H, pk, \hat{y}, x, \ell, c, s), \tau)$ and $\mathsf{SIG.Verify}_{vk}((\ell, c), \sigma)$ accept and if yes output $\mathsf{FHE.Dec}_{sk}(\hat{y})$, otherwise output $\perp$. Increase cntr by 1, shut down if $\mathsf{cntr} = n_{\mathsf{invoc}}$.

4.1 Adaptive Soundness

We now turn to prove adaptive soundness of Construction 8. First, observe that we do not need to use the sophisticated construction for collision resistant hash-functions based on error-correcting codes used in [BG02]. [BG02] need this construction since an efficient corrupted prover might only be *implicitly* aware of a hash-collision of super-polynomial length. However, to extract such a hash-collision, an extractor would have to run in super-polynomial time (this was the approach taken in [Bar01]). However, this problem does not occur in the adaptive soundness experiment. A corrupted prover first has to demonstrate explicit awareness of the statement (which has the same length as the witness). Thus, the size of the witness that needs to be extracted is a-priori bounded by a fixed polynomial depending on the corrupted prover. Thus the extractor runs efficiently.

Theorem 1. *The argument-system* $(\mathsf{CRSGen}, \mathsf{P}', \mathsf{V}')$ *in Construction 8 is adaptively sound, provided that the argument system* (P, V) *is sound and the family of hash-functions* $\{H_i\}$ *is collision-resistant.*

We will split Theorem 1 in two lemmata establishing the indistinguishability between the following two experiments. Let P^* be an arbitrary PPT-prover as in the definition of the adaptive soundness experiment.

Experiment 1 This experiment is identical to the adaptive soundness experiment.

Experiment 2 Identical to experiment 1, except that this experiment only outputs 1 if additionally to $\langle \mathsf{P}^*, \mathsf{V}' \rangle (H, (k, pk, \hat{y}, x, \tilde{M}, s)) = 1$ and $(k, pk, \hat{y}, x, \tilde{M}, s) \notin L'$ it holds that $(k, H, pk, \hat{y}, x, |\tilde{M}|, H(\tilde{M}, s)) \in L$.

Denote the output of experiment 1 by $\mathbf{Exp}_1(\mathsf{P}^*)$ and the output of experiment 2 by $\mathbf{Exp}_2(\mathsf{P}^*)$.

Lemma 9. *Given that the argument-system* (P, V) *is sound, it holds that* $\Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1] - \Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] \leq \mathsf{negl}(k)$

Proof. We will show the contraposition. Assume that $\Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1] > \Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] + \epsilon$. We will construct a corrupted prover $\mathsf{P}^\dagger$ that breaks the soundness-property of (P, V) with advantage ϵ. Let $x' = (k, H, pk, \hat{y}, x, |\tilde{M}|, H(\tilde{M}), s)$ and $x' = (k, pk, \hat{y}, x, \tilde{M}, s)$.

First notice that $\Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] = \Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1 \text{ and } x \in L] = \Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1 \text{ and } x \in L]$. It holds that $\Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1 \text{ and } x \notin L] > \epsilon$, since

$$\begin{aligned}
\epsilon &< \Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1] - \Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] \\
&= \Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1 \text{ and } x \notin L] + \Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1 \text{ and } x \in L] - \Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] \\
&= \Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1 \text{ and } x \notin L]
\end{aligned}$$

As it holds that $\Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1 \text{ and } x \notin L] > \epsilon$, a simple averaging argument shows that there must exist a fixed H and a fixed random-tape r for P^* such that the same holds. We can therefore fix H and r. Let P^*'s outputs be a statement $x' = (k, pk, \hat{y}, x, \tilde{M}, s)$, which we can assume to be fixed. We will now show that $\mathsf{P}^\dagger$ breaks the soundness of (P, V) for the statement $x = (k, H, pk, \hat{y}, x, |\tilde{M}|, H(\tilde{M}), s)$. The prover $\mathsf{P}^\dagger$ then simulates the interaction of P^* with V' by forwarding the messages between P^* and the verifier V he interacts with. We claim that $\Pr[\langle \mathsf{P}^\dagger, \mathsf{V} \rangle(x) = 1] > \epsilon$. Observe in $\mathbf{Exp}_1$ and the simulation of $\mathsf{P}^\dagger$ the views of P^* are identically distributed. Thus, the probability that V accepts is also the same in both experiments. Hence

$$\begin{aligned}
\Pr[\langle \mathsf{P}^\dagger, \mathsf{V} \rangle = 1 \text{ and } x \notin L] &= \Pr[\langle \mathsf{P}^*, \mathsf{V}' \rangle(x') = 1 \text{ and } x \notin L] \\
&= \Pr[\mathbf{Exp}_1(\mathsf{P}^*) = 1 \text{ and } x \notin L] > \epsilon
\end{aligned}$$

which means that $\mathsf{P}^\dagger$ breaks the soundness of (P, V) with advantage ϵ. □

Lemma 10. *Given that the family of hash-functions $\{H_i\}$ is collision resistant, then it holds that* $\Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] \leq \mathsf{negl}(k)$

Proof. Again we show the contrapositive, thus assume that $\Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] > \epsilon$. If $\mathbf{Exp}_2$ outputs 1, it holds that $(k, pk, \hat{y}, x, \tilde{M}, s) \notin L'$ but $(k, H, pk, \hat{y}, x, |\tilde{M}|, H(\tilde{M}), s) \in L$. Thus there exists an $\tilde{M}'$ with $H(\tilde{M}') = H(\tilde{M})$ and $|\tilde{M}'| = |\tilde{M}|$ such that $(k, pk, \hat{y}, x, \tilde{M}', s) \in L$. This necessarily implies that $\tilde{M} \neq \tilde{M}'$. Let Ext be a knowledge extractor for (P, V). Ext runs in time polynomial in the size of the witness $\tilde{M}'$, which has the same size as $\tilde{M}$. As the adaptive soundness experiment requires P^* to output a statement containing $\tilde{M}$ explicitly, the size of the witness $\tilde{M}'$ is upper bounded by a fixed polynomial (e.g. runtime of P^*). Thus the extractor Ext runs in polynomial time. We construct an adversary $\mathcal{A}$ against the collision-resistance of $\{H_i\}$. $\mathcal{A}$'s input is a hash-function H sampled from $\{H_i\}$. $\mathcal{A}$ first chooses random tapes for the machines P^* and V. It runs P^* on input H until P^* outputs a statement $(k, pk, \hat{y}, x, \tilde{M}, s)$. Then $\mathcal{A}$ provides V with input $(k, H, pk, \hat{y}, x, |\tilde{M}|, H(\tilde{M}), s)$ and simulates interaction between P^* and V^*. The assumption $\Pr[\mathbf{Exp}_2(\mathsf{P}^*) = 1] > \epsilon$ yields that V accepts with probability greater than ϵ. If V accepts, $\mathcal{A}$ runs the extractor Ext on P^* to obtain a witness $\tilde{M}'$. The extraction has non-negligible success probability, and thus $\mathcal{A}$ can output a hash-collision $(\tilde{M}, \tilde{M}')$ with non-negligible probability. □

5 From Conditional Decryption to Obfuscation

We are now ready to provide a construction of a composable obfuscation-scheme based on conditional decryption. The idea of the protocol can be outlined as follows. The sender S encodes his

input-programs M as inputs for universal circuits, universal oblivious Turing-machines or a similar deterministic input-oblivious model of computation. Such programs M are then encrypted under a fully homomorphic encryption scheme FHE. To allow the receiver to decrypt evaluation results of encrypted circuits, the sender instantiates a conditional decrypt functionality $\mathcal{F}_{\mathsf{CONDEC}}$ with a secret key sk for FHE and a verification-key vk for SIG. To allow the receiver to use this conditional decryption oracle, S sends encrypted programs $\tilde{M}$ together with a hash-value $c = H(\tilde{M})$ and a signature σ on c. The purpose of the signature is to make sure that the encrypted program $\tilde{M}$ was issued by the S and not manipulated by R. R can evaluate an obfuscation of a program M as follows. First Rqueries $\mathcal{F}_{\mathsf{CONDEC}}$ for random coins to use for the homomorphic evaluation of $\hat{M}$. The coins are provided by $\mathcal{F}_{\mathsf{CONDEC}}$, as a corrupted R might use coins that lead to an decryption error. Then, R homomorphically evaluates $\tilde{M}$ on her input x using said coins. She obtains a the plaintext of the evaluation result $M(x)$ by using $\mathcal{F}_{\mathsf{CONDEC}}$. We will now provide a formal description of our protocol Π_{Obf}. Let SIG be an EUF-CMA secure signature-scheme, FHE be an IND-CPA secure fully homomorphic encryption scheme, $\{F_i\}$ a family of pseudorandom functions and $\{H_i\}$ a family of hash-functions as above.

Protocol Π_{Obf}

- Setup
  Sender S:
  - Compute $(pk, sk) \leftarrow \mathsf{FHE.KeyGen}(1^k)$, $(vk, sgk) \leftarrow \mathsf{SIG.KeyGen}(1^k)$, sample $F \leftarrow \{F_i\}$, $H \leftarrow_R \{H_i\}$ and choose private coins $\mathsf{state}_0 \leftarrow_R \{0,1\}^k$ for the verifier V.
  - Input $(\mathsf{state}_0, F, H, vk, sk)$ to $\mathcal{F}_{\mathsf{CONDEC}}$.
  - Send (H, pk) to R.

  Receiver R:
  - Upon receiving a tuple (H, pk), store it.
- $\mathsf{Obfuscation}$
  Sender S:
  - Upon receiving input $(\mathtt{Obfuscate}, \mathtt{id}, M)$ for a program M, check if $\mathtt{id}$ has already been stored. If not, compute $\hat{M} \leftarrow \mathsf{FHE.Enc}_{pk}(M)$ and set $\tilde{M} = (\hat{M}.\mathsf{spec}_M)$. Compute $c \leftarrow H(\tilde{M})$, $\ell = |\tilde{M}|$ and $\sigma \leftarrow \mathsf{SIG.Sign}_{sgk}(\ell, c)$. Send $(\mathtt{id}, \tilde{M}, \ell, c, \sigma)$ to R.

  Receiver R:
  - Upon receiving a tuple $(\mathtt{id}, \tilde{M}, \ell, c, \sigma)$, store it.
- Query (only Receiver R)
  - Upon receiving input $(\mathtt{Query}, \mathtt{id}, x)$, check if a tuple $(id, \tilde{M}, \ell, c, \sigma)$ has been stored. If not, abort and output $\perp$.
  - Input $(\mathtt{coins}, x, c)$ into $\mathcal{F}_{\mathsf{CONDEC}}$ and let s be the corresponding output.
  - Compute $\hat{y} \leftarrow \mathsf{FHE.Evaluate}_{pk}(\tilde{M}, x; r)$.

– Setup a prover P with input $(k, H, pk, \hat{y}, x, \ell, c, s)$ and witness-input $(\tilde{M})$. Let P interact with the verifier V in $\mathcal{F}_{\mathsf{CONDEC}}$ by forwarding messages between $\mathcal{F}_{\mathsf{CONDEC}}$ and P (converting messages sent by P into `Verifier`-messages for $\mathcal{F}_{\mathsf{CONDEC}}$). Stop the simulation once V accepts. Let τ be the last message sent by P.

– Input $(\texttt{Decrypt}, (\hat{y}, x, \ell, c, \sigma, \tau))$ into $\mathcal{F}_{\mathsf{CONDEC}}$ and let y be the output of $\mathcal{F}_{\mathsf{CONDEC}}$.

– Output y.

5.1 Proof of Security

5.1.1 Corrupted Sender

We first prove perfect UC-security in case of a corrupted sender, as this is the easy case. Let $\mathcal{A}_{\mathsf{S}}$ be the dummy-adversary for a corrupted sender. We will construct a simulator $\mathcal{S}_{\mathsf{S}}$. Since the protocol Π_{Obf} is non-interactive (messages are only passed from the S to R) and an honest R is deterministic, the simulator's task is rather simple. Let $(\mathsf{state}_0, H, vk, sk)$ be $\mathcal{A}_{\mathsf{S}}$'s inputs to $\mathcal{F}_{\mathsf{CONDEC}}$, with additional adversarial input n_{invoc}. $\mathcal{S}_{\mathsf{S}}$ first inputs n_{invoc} to $\mathcal{F}_{\mathsf{Obf}}$. Every time $\mathcal{A}_{\mathsf{S}}$ sends a new message $(\texttt{id}, \tilde{M}, \ell, c, \sigma)$ to R, $\mathcal{S}_{\mathsf{S}}$ constructs a program M that does the following. M takes an input x and simulates the interaction between a receiver-machine R and a conditional decryption machine $\mathcal{F}_{\mathsf{CONDEC}}$, where R gets input x and $\mathcal{F}_{\mathsf{CONDEC}}$ gets input $(\mathsf{state}_0, H, vk, sk)$. M outputs whatever R outputs. For any environment $\mathcal{Z}$, the indistinguishability of $\mathsf{Real}^{\mathcal{A}_{\mathsf{S}}}_{\Pi_{\mathsf{Obf}}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{S}}}_{\mathcal{F}_{\mathsf{Obf}}}(\mathcal{Z})$ follows by combining machines accordingly

5.1.2 Corrupted Receiver

We will now prove computational UC-security in case of a corrupted receiver. We will first state the simulator $\mathcal{S}_{\mathsf{R}}$ against a corrupted receiver $\mathcal{A}_{\mathsf{R}}$, which is *w.l.o.g.* the dummy adversary.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Prepare a simulated sender-machine S and run the setup-phase of S. Send S's output (H, pk) to $\mathcal{A}_{\mathsf{R}}$. Setup a simulated $\mathcal{F}_{\mathsf{CONDEC}}$ and send the `ready` message from $\mathcal{F}_{\mathsf{CONDEC}}$ to $\mathcal{A}_{\mathsf{R}}$.
- Upon receiving a message $(\texttt{obfuscation}, \texttt{id}, \mathsf{spec}_M)$ from $\mathcal{F}_{\mathrm{Obf}}$, set $M \leftarrow 0^*$ to be the all-zero program with specification spec_M, input $(\texttt{obfuscate}, \texttt{id}, M)$ to the simulated sender S and run it. Let $m = (\texttt{id}, \tilde{M}, \ell, c, \sigma)$ be the output of S. Send m to $\mathcal{A}_{\mathsf{R}}$ and store m.
- Upon receiving a message $m = (\texttt{coins}, (x, \ell, c))$ from $\mathcal{A}_{\mathsf{R}}$, reply with random coins r and store the tuple (m, r) to reply subsequent identical calls (simulated random oracle).
- Upon receiving a message $m = (\texttt{Verifier}, (\hat{y}, x, \ell, c, r, \tau))$ from $\mathcal{A}_{\mathsf{R}}$, forward m to the simulated $\mathcal{F}_{\mathsf{CONDEC}}$ and forward the corresponding output to $\mathcal{A}_{\mathsf{R}}$.
- Upon receiving a message $m = (\texttt{Decrypt}, (\hat{y}, x, \ell, c, r, \sigma, \tau))$ from $\mathcal{A}_{\mathsf{R}}$, check if $(\texttt{id}, \tilde{M}, \ell, c, r, \sigma)$ has been stored, if not output $\perp$ to $\mathcal{A}_{\mathsf{R}}$. Check if it holds that $\hat{y} = \mathsf{FHE.Evaluate}_{pk}(\tilde{M}, x; r)$ and $\mathcal{F}_{\mathsf{CONDEC}}$ accepts message m. If not output $\perp$. Otherwise input $(\texttt{Query}, \texttt{id}, x)$ into $\mathcal{F}_{\mathsf{Obf}}$ and let y be the output. Output y to $\mathcal{A}_{\mathsf{R}}$.

Now we will show that for every PPT-environment $\mathcal{Z}$, $\mathsf{Real}^{\mathcal{A}_\mathsf{R}}_{\Pi_\mathsf{Obf}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_\mathsf{R}}_{\mathcal{F}_\mathsf{Obf}}(\mathcal{Z})$ are computationally indistinguishable. Consider the following sequence of hybrid experiments. In Experiment i the environment $\mathcal{Z}$ interacts with a simulator $\mathcal{S}_i$.

Experiment 1 In this experiment, simulator $\mathcal{S}_1$ simulates the real protocol Π_Obf.

Experiment 2 Identical to experiment 1, except that $\mathcal{S}_2$ aborts if the event happens that $\mathcal{A}_\mathsf{R}$ sends a message $(\mathtt{Decrypt}, (\hat{y}, x, \ell, c, \sigma, \tau))$ to $\mathcal{F}_\mathsf{CONDEC}$ such that it holds $\mathsf{SIG.Verify}_{vk}((\ell, c), \sigma) = 1$ but the signature σ has not been generated by $\mathcal{S}_2$.

Experiment 3 Identical to experiment 2, except that now $\mathcal{S}_3$ aborts if the event happens that $\mathcal{A}_\mathsf{R}$ sends a message $(\mathtt{Decrypt}, (\hat{y}, x, \ell, c, \sigma, \tau))$ to $\mathcal{F}_\mathsf{CONDEC}$ and V accepts, even though it does not hold that $\hat{y} = \mathsf{FHE.Evaluate}_{pk}(\tilde{M}, x; 0^*)$.

Experiment 4 Identical to experiment 3, except that the `coins`-calls to $\mathcal{F}_\mathsf{CONDEC}$ are replied with truly random coins.

Experiment 5 Identical to experiment 4, except that now y is computed by $y \leftarrow M(x)$, where M is the program that corresponds to (ℓ, c).

Experiment 6 Identical to experiment 5, except that now, once the simulator $\mathcal{S}_5$ gets input $(\mathtt{obfuscate}, \mathtt{id}, M)$ the simulator chooses the string M' to be the all-zero program with the same specification spec_M as M and computes $\hat{M} \leftarrow \mathsf{FHE.Enc}_{pk}(M')$ instead of $\hat{M} \leftarrow \mathsf{FHE.Enc}_{pk}(M)$. This is the ideal experiment.

Remarks A few remarks are in place. While in experiment 1 it is possible that the conditional decrypt functionality answers decryption queries that include tuples (ℓ, c) that were not signed by the sender, this is impossible in experiment 2. Computational indistinguishability of experiments 1 and 2 is established by the existential unforgeability of the signature scheme SIG. In experiment 3 the simulator aborts if $\mathcal{A}_\mathsf{R}$ manages to convince $\mathcal{F}_\mathsf{CONDEC}$ of a false statement. Note that an efficient simulator can decide whether $\hat{y} = \mathsf{FHE.Evaluate}_{pk}(\tilde{M}, x; 0^*)$. Computational indistinguishability is established using the adaptive computational soundness property of the argument-system $(\mathsf{CRSGen}, \mathsf{P}', \mathsf{V}')$. In experiment 4 the coins output by $\mathcal{F}_\mathsf{CONDEC}$ are truly random instead of pseudorandom. Computational indistinguishability follows by the pseudorandomness-property of the family $\{F_i\}$. In experiment 5 the outputs of $\mathcal{F}_\mathsf{CONDEC}$ are determined by evaluating the program M on input x directly. Statistical indistinguishability follows from the correctness of FHE. In experiment 6, encrypted programs $\hat{M}$ are replaced by encryptions $\hat{M}'$ of all-zero programs. The IND-CPA security of the fully homomorphic encryption scheme FHE implies computational indistinguishability.

Lemma 11. *From $\mathcal{Z}$'s view, experiment 1 and experiment 2 are computationally indistinguishable, given that SIG is a existentially unforgeable signature scheme.*

Proof. Unless $\mathcal{S}_2$ aborts, $\mathcal{Z}$'s view is identically distributed in experiment 1 and experiment 2. Thus a $\mathcal{Z}$ distinguishing between experiment 1 and experiment 2 must be able to provoke an

abort with non-negligible probability. Assume there exists a PPT-environment $\mathcal{Z}$ and a non-negligible $\epsilon(k)$ such that $\Pr[\mathcal{S}_2 \text{ aborts interacting with } \mathcal{Z}] \geq \epsilon(k)$, where the probability is taken over the coins of $\mathcal{S}_2$ and $\mathcal{Z}$. We now construct an EUF-CMA adversary $\mathcal{B}$ for the signature-scheme SIG, i.e. $\mathcal{B}$ receives a signature-key vk^* from the EUF-CMA experiment, has access to a signature-oracle and wins if it succeeds in existentially forging a signature σ^* for a message of its choice m^*. $\mathcal{B}$ does the same as $\mathcal{S}_2$, except that it does not generate vk itself but sets $vk \leftarrow vk^*$. Moreover, instead of signing the tuples (ℓ, c) using the signature-key sk, it sends signature queries c to its signature-oracle to get signatures σ. Finally, after an abort is decided because it holds that $\mathsf{SIG.Verify}_{vk}((\ell, c), \sigma) = 1$, even though (ℓ, c) has not been signed by the sender, $\mathcal{B}$ sets $m^* \leftarrow (\ell, c)$, $\sigma^* \leftarrow \sigma$ and returns (m^*, σ^*) to the EUF-CMA experiment. Obviously, $\mathcal{Z}$'s views are identically distributed when interacting with either $\mathcal{S}_2$ or $\mathcal{B}$ and thus it holds that $\Pr[\mathcal{B} \text{ existentially forges a signature}] = \Pr[\mathcal{S}_2 \text{ aborts interacting with } \mathcal{Z}] \geq \epsilon(n)$, which contradicts the existential unforgeability of SIG. □

Lemma 12. *From $\mathcal{Z}$'s view, experiment 2 and experiment 3 are computationally indistinguishable, given that the adaptive resettable soundness-property of the argument-system* $(\mathsf{CRSGen}, \mathsf{P}', \mathsf{V}')$ *holds.*

Proof. Assume there exists a PPT-environment $\mathcal{Z}$ that distinguishes between experiment 4 and experiment 5 with non-negligible advantage $\epsilon(k)$. But this means that with probability $\epsilon(k)$ $\mathcal{A}_{\mathsf{R}}$ is able to convince $\mathcal{F}_{\mathsf{CONDEC}}$ of a false statement $\hat{y} = \mathsf{FHE.Evaluate}_{pk}(\tilde{M}, x; 0^*)$. Thus it holds that $\Pr[\mathcal{S}_3 \text{ aborts in decrypt-call }] \geq \epsilon(k)$. We will now construct an adaptive corrupted prover P^* that breaks the adaptive resettable soundness property of $(\mathsf{CRSGen}, \mathsf{P}', \mathsf{V}')$. Let $q(k) = \mathsf{poly}(k)$ be an upper bound on the number of different statements $\mathcal{A}_{\mathsf{R}}$ tries to prove to $\mathcal{F}_{\mathsf{CONDEC}}$. P^* first guesses an index $i \in \{1, \ldots, q(k)\}$ of a statement for which $\mathcal{Z}$ provokes $\mathcal{S}_3$ to abort. Clearly, the probability that P^* guesses the right index i is $\geq 1/q(k)$. P^* will answer all queries of $\mathcal{A}_{\mathsf{R}}$ that do not refer to the i-th statement like $\mathcal{S}_3$, using the simulated verifier of $\mathcal{F}_{\mathsf{CONDEC}}$. However, once $\mathcal{A}_{\mathsf{R}}$ sends a query with the i-th different statement $(\hat{y}_i, x_i, \tilde{M}_i)$ for the first time, P^* announces the statement $(pk, \hat{y}_i, x_i, \tilde{M}_i)$ to the adaptive soundness experiment. From now on, P^* forwards all messages from $\mathcal{A}_{\mathsf{R}}$ to $\mathcal{F}_{\mathsf{CONDEC}}$ that involve the statement $(\hat{y}_i, x_i, \tilde{M}_i)$ to the experiment, and uses the verifier-messages it receives from the experiment as verifier-messages of $\mathcal{F}_{\mathsf{CONDEC}}$. Given that the coins returned by the verifier are computationally indistinguishable from uniformly random, an efficient $\mathcal{Z}$ will not notice a difference when interacting with either $\mathcal{S}_3$ or P^*. It holds that $\Pr[\mathsf{P}^* \text{ aborts in decrypt-call }] \geq \Pr[\mathcal{S}_3 \text{ aborts in decrypt-call }] - \mathsf{negl}(k) \geq \epsilon'(k)$ for a non-negligible $\epsilon'(k)$. Thus we have that

$$\begin{aligned}\Pr[\mathsf{P}^* \text{ convinces } \mathsf{V}' \text{ of false statement}] &= \Pr[\mathsf{P}^* \text{ guesses the right } i \text{ and aborts in decrypt call}] \\ &\geq \epsilon'(k)/q(k)\end{aligned}$$

which is non-negligible in k. □

Lemma 13. *From $\mathcal{Z}$'s view, experiment 3 and experiment 4 are computationally indistinguishable, given that $\{F_i\}$ is a family of pseudorandom functions*

Lemma 14. *From $\mathcal{Z}$'s view, experiment 4 and experiment 5 are computationally indistinguishable, given that the fully homomorphic encryption-scheme FHE is correct against PPT-adversaries.*

Proof. Conditioned to the event that no decryption-error occurs, experiment 4 and experiment 5 are perfectly indistinguishable. Assume there was an environment $\mathcal{Z}$ that can provoke decryption-errors with non-negligible advantage ϵ. We will now construct a PPT-adversary $\mathcal{B}$ that breaks the correctness-property of FHE. Assume that the corrupted receiver R makes at most $p(k) = \mathsf{poly}(k)$ `coins`-calls to $\mathcal{F}_{\mathsf{CONDEC}}$. $\mathcal{B}$ first guesses an index $i \in \{1, \ldots, p(k)\}$ for which a decryption-error will occur. $\mathcal{B}$ then simulates $\mathcal{S}_4$ using the public-key pk and the private key sk provided by the COR-experiment. Let (x, ℓ, c) by the i-th `coins`-call made by R, where $c = H(\hat{M})$ for a encrypted program $\hat{M}$. $\mathcal{B}$ then outputs $(\hat{M}, x)$. Now, the probability that a decryption-error occurs is $> \epsilon$. Let i_0 be the index of a `coins`-call for which a decryption-error occurs. The probability that $\mathcal{B}$ guesses i_0 is $\geq p(k)$. Therefore, the probability that a decryption-error occurs at the index i guessed by $\mathcal{B}$ is $> \epsilon/p(k)$, which is non-negligible. □

Lemma 15. *From $\mathcal{Z}$'s view, experiment 5 and experiment 6 are computationally indistinguishable, given that FHE is a IND-CPA secure fully homomorphic encryption scheme.*

Proof. Assume that $\mathcal{Z}$ distinguishes between experiment 5 and experiment 6 with non-negligible advantage $\epsilon(k)$. We will construct a IND-CPA adversary $\mathcal{B}$ that breaks the IND-CPA security of FHE with non-negligible advantage. We will employ a hybrid-argument in this proof. Let $p(k) = \mathsf{poly}(k)$ be an upper bound for the number of sender-inputs $(\texttt{obfuscate}, id, M)$ that $\mathcal{Z}$ provides. $\mathcal{B}$ first chooses a $t \in \{1, \ldots, p(k)\}$ uniformly at random. $\mathcal{B}$ then does exactly the same as $\mathcal{S}_5$, except for the following. Instead of generating the public key pk itself, $\mathcal{B}$ sets $pk \leftarrow pk^*$, where pk^* is the public key that was provided to $\mathcal{B}$ by the IND-CPA experiment. For all sender-inputs $(\texttt{obfuscate}, id, M_i)$ of $\mathcal{Z}$ with $i < t$, $\mathcal{B}$ computes $\hat{M}_i \leftarrow \mathsf{FHE.Enc}_{pk}(M_i)$, like $\mathcal{S}_5$ does. For the t-th input $(\texttt{obfuscate}, id, M_t)$, $\mathcal{B}$ sends the challenge-messages M_t and M'_t to the IND-CPA experiment, where M'_t is an all-zero program with the same specification as M_t. $\mathcal{B}$ receives a challenge-ciphertext $\hat{M}^*$ from the experiment and sets $\hat{M}_t \leftarrow \hat{M}^*$. For all further inputs $(\texttt{obfuscate}, id, M_i)$ with $i > u$, $\mathcal{B}$ chooses M'_i to be the all-zero program with the same specification as M_i and sets $\hat{M}_i \leftarrow \mathsf{FHE.Enc}_{pk}(M'_i)$, like $\mathcal{S}_6$. When the simulation terminates, $\mathcal{B}$ outputs whatever $\mathcal{Z}$ outputs.

Let $u \in \{0, 1\}$ be the secret coin of the IND-CPA experiment that decides whether M_t or a random M'_t is encrypted in $\hat{M}$. We will calculate $\Pr[\mathcal{B} = 1 | u = 0]$ and $\Pr[\mathcal{B} = 1 | u = 1]$. It holds that

$$\Pr[\mathcal{B} = 1 | u = 0] = \sum_{i=1}^{p(k)} \frac{1}{p(k)} \Pr[\mathcal{B} = 1 | t = i, u = 0]$$

$$\Pr[\mathcal{B} = 1 | u = 1] = \sum_{i=1}^{p(k)} \frac{1}{p(k)} \Pr[\mathcal{B} = 1 | t = i, u = 1]$$

Observing that for $i = 1, \ldots, p(k) - 1$ the view of $\mathcal{Z}$ is identically distributed for either $t = i$ and $u = 0$ or $t = i + 1$ and $u = 1$, we get that

$$\Pr[\mathcal{B} = 1 | t = i, u = 0] = \Pr[\mathcal{B} = 1 | t = i + 1, u = 1].$$

thus it holds that

$$|\Pr[\mathcal{B}=1|u=0]-\Pr[\mathcal{B}=1|u=1]| = \frac{1}{p(k)}\,|\Pr[\mathcal{B}=1|t=p(k),u=0]-\Pr[\mathcal{B}=1|t=1,u=1]|$$
$$= \frac{1}{p(k)}\left|\Pr[\mathcal{Z}^{\mathcal{S}_4}=1]-\Pr[\mathcal{Z}^{\mathcal{S}_5}=1]\right| = \frac{\epsilon(k)}{p(k)}$$

Hence $\mathcal{B}$ distinguishes $u=0$ from $u=1$ with non-negligible advantage and thus breaks the IND-CPA security of FHE, contradicting the assumption. □

6 Implementing Conditional Decryption with Stateless Hardware

Next we will provide a protocol that UC-implements the CONDEC-functionality with a stateless hardware token. To provide additional security guarantees, we allow a corrupted sender to create malicious stateful hardware tokens. As the isolation between the environment and the token is essential for receiver-security, we need to eliminate possible narrow-band channels between the two. We therefore need to make two restrictions to the class of programs that can be obfuscated. We restrict to programs that have both a fixed input-size and output-size, that are polynomial in the security parameter. If we allowed inputs or outputs of arbitrary length, the environment could encode messages for the token in the length of the inputs or outputs. We need to make all calls of R to $\mathcal{F}_{\mathsf{CONDEC}}$ indistinguishable for a malicious token T^*. This means in particular that all messages from R to the token T need to have the same length. As the length of the inputs x and outputs y are fixed, and all other messages that R sends to $\mathcal{F}_{\mathsf{CONDEC}}$ have also a fixed length independent of M, we can use a padding to pad all messages to the maximum length. A straightforward approach to implement $\mathcal{F}_{\mathsf{CONDEC}}$ with a stateless token would be to run a general resettable multiparty-computation protocol between the receiver and the token. However, a general approach might increase the round-complexity of a decrypt-call, which is so far constant. Since we already require a fully-homomorphic encryption scheme for the construction of protocol Π_{Obf}, we can use an efficient two-round protocol based on fully homomorphic encryption and NIZKPoKs to implement $\mathcal{F}_{\mathsf{CONDEC}}$. Our protocol will require two common reference strings. We first give an outline of the protocol Π_{CONDEC}. Let CRS_1 and CRS_2 be two reusable common reference strings and let FHE be a fully homomorphic encryption scheme. Let the input of S be a. In the setup phase, the sender first computes a commitment $c \leftarrow \mathsf{com}(a;r)$, using a non-interactive perfectly binding commitment-scheme com. The sender S creates a stateless token T with the following functionality. T gets messages that consist of a public key pk, a ciphertext $\hat{m}$ and a NIZKPoK π_1 with respect to CRS_1. The proof π_1 states that $\hat{m}$ is a proper encryption of a query to $\mathcal{F}_{\mathsf{CONDEC}}$ under the public key pk. If π_1 is valid, T generates random coins r', r'' by applying a pseudorandom function to $(pk, \hat{m}, \pi_1)$. Using the coins r', T computes $\hat{s} \leftarrow \mathsf{FHE.Evaluate}_{pk}(\mathcal{F}_{\mathsf{CONDEC}}, a, \hat{m}; r')$. Finally, T computes, using CRS_2 and coins r'', a NIZKPoK π_2 which states that for the tuple $(pk, c, \hat{m}, \hat{s})$ it holds that there exists an input a and coins r such that $c = \mathsf{com}(a;r)$ and $\hat{s} = \mathsf{FHE.Evaluate}_{pk}(\mathcal{F}_{\mathsf{CONDEC}}, a, \hat{m}; r')$. T then outputs $(\hat{s}, \pi_2)$. This concludes the description of the token. S sends the wrapped token T together with the commitment c to R. Upon receiving the commitment c and the wrapped token T, the receiver R stores c and generates a pair of public and secret key (pk, sk) for FHE. We now give a description of the receiver R in the query-phase. Let m be an input of R. R first encrypts m under the public key pk and obtains a ciphertext $\hat{m}$. R then uses CRS_1 to compute a NIZKPoK π_1 that $\hat{m}$ is a valid ciphertext. R sends $\hat{m}$ together with the proof π_1 to T and receives an output $(\hat{s}, \pi_2)$.

If the proof π_2 is valid for the statement $(pk, c, \hat{m}, \hat{s})$ with CRS_2, then R decrypts $s \leftarrow \mathsf{FHE.Dec}_{sk}(\hat{s})$ and outputs s. Finally, in order to plug our last protocol Π_{CRS} for the offline initialization-phase into protocol Π_{CONDEC}. We require S to issue the token to R *before* the common reference strings are computed (as the token will be involved in the generation of the common reference strings). To provide the token with common reference strings for the NIZKPoKs, the sender authenticates them with a signature.

We will now provide the full construction of protocol Π_{CONDEC} and prove UC-security. Let $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ be the wrapper functionality for the stateless hardware, FHE be a fully homomorphic encryption scheme, SIG be an EUF-CMA secure signature scheme, com be a non-interactive perfectly binding commitment scheme, CRS_1 and CRS_2 be two reusable common reference strings. We define the languages $L_1 = \{(pk, \hat{m}) : \exists (m, r) \text{ s.t. } \hat{m} = \mathsf{FHE.Enc}_{pk}(m; r)\}$ and $L_2 = \{(pk, c, \hat{m}, \hat{s}) : \exists (a, r', r'') \text{ s.t. } c = \mathsf{com}(a; r') \text{ and } \hat{s} = \mathsf{FHE.Evaluate}_{pk}(\mathcal{F}_{\mathsf{CONDEC}}, a, \hat{m}; r'')\}$. Let $(\mathsf{P}_1, \mathsf{V}_1)$ be a NIZKPoK-system for L_1 and $(\mathsf{P}_2, \mathsf{V}_2)$ be a NIZKPoK-system for L_2. Let $\{F_i\}$ be a family of pseudorandom functions with appropriate range.

Protocol Π_{CONDEC}

- Trusted Setup
  - Set $\mathsf{CRS}_1 \leftarrow \mathsf{CRSGen}_1(1^k)$
  - Set $\mathsf{CRS}_2 \leftarrow \mathsf{CRSGen}_2(1^k)$
- Setup
  Sender S:
  - Upon receiving input $(\mathtt{Setup}, a)$, compute $c \leftarrow \mathsf{com}(a; r)$ with fresh coins r, sample an $F \leftarrow_R \{F_i\}$ and generate $(sgk, vk) \leftarrow \mathsf{SIG.KeyGen}(1^k)$.
  - Setup a token T with the following specification.
    Token T:
    * Upon receiving input $((\mathsf{CRS}_1, \mathsf{CRS}_2), \sigma, pk, \hat{m}, \pi_1)$, check if $\mathsf{SIG.Verify}_{vk}((\mathsf{CRS}_1, \mathsf{CRS}_2), \sigma)$ and $\mathsf{V}_1(\mathsf{CRS}_1, (pk, \hat{m}), \pi_1) = 1$, if not abort.
    * Set $(r', r'') \leftarrow F(pk, \hat{m}, \pi_1)$.
    * Compute $\hat{s} \leftarrow \mathsf{FHE.Evaluate}_{pk}(\mathcal{F}_{\mathsf{CONDEC}}, a, \hat{m}; r')$.
    * Compute $\pi_2 \leftarrow \mathsf{P}_2(\mathsf{CRS}_2, (pk, c, \hat{m}, \hat{s}), (a, r, r'); r'')$.
    * Output $(\hat{s}, \pi_2)$
  - Send c, σ to R and input T into $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$.

  Receiver R:
  - Upon receiving a message (c, σ), store (c, σ) and compute $(pk, sk) \leftarrow \mathsf{FHE.KeyGen}(1^k)$.
- Queries
  Receiver R:
  - Upon receiving a message $m = (\mathtt{Verifier}, m')$ or $m = (\mathtt{Decrypt}, m')$, pad m to maximum-length of messages that $\mathcal{F}_{\mathsf{CONDEC}}$ takes for security parameter k.

– Compute $\hat{m} \leftarrow \mathsf{FHE.Enc}_{pk}(m; r)$. Compute $\pi_1 \leftarrow \mathsf{P}_1(\mathsf{CRS}_1, (pk, \hat{m}), (m, r))$.
– Input $((\mathsf{CRS}_1, \mathsf{CRS}_2), \sigma, pk, \hat{m}, \pi_1)$ into $\mathcal{F}^{\mathsf{wrap}}_{\mathsf{stateless}}$. Let $(\hat{s}, \pi_2)$ be the corresponding output.
– Check if $\mathsf{V}_2(\mathsf{CRS}_2, (pk, c, \hat{m}, \hat{s}), \pi_2) = 1$, if so output $s = \mathsf{FHE.Dec}_{sk}(\hat{s})$, otherwise abort.

6.1 Proof of Security

Let Sim_1 and Ext_1 be the simulator and extractor for $(\mathsf{CRSGen}_1, \mathsf{P}_1, \mathsf{V}_1)$ and Sim_2 and Ext_2 the same for $(\mathsf{CRSGen}_2, \mathsf{P}_2, \mathsf{V}_2)$ respectively. Let $\mathcal{R}_{L_1}$ and $\mathcal{R}_{L_2}$ be the witness-relations for L_1 and L_2

6.1.1 Corrupted Receiver

We will now prove computational UC-security against a corrupted receiver. Let $\mathcal{A}_{\mathsf{R}}$ be the corrupted receiver, which is *w.l.o.g.* the dummy adversary. We first state the simulator $\mathcal{S}_{\mathsf{R}}$

Simulator $\mathcal{S}_{\mathsf{R}}$

- Set $(\mathsf{CRS}_1, td_1) \leftarrow \mathsf{FakeGen}_1(1^k)$ and $(\mathsf{CRS}_2, td_2) \leftarrow \mathsf{FakeGen}_2(1^k)$.
- Simulate the setup-phase of a honest sender S with random fake input a' and send the message (c, σ) to $\mathcal{A}_{\mathsf{R}}$. Let T be the token output be S. Let H be a random oracle with the same size as the pseudorandom function F.
- Upon receiving a message $((\mathsf{CRS}_1, \mathsf{CRS}_2), \sigma, pk, \hat{m}, \pi_1)$ from $\mathcal{A}_{\mathsf{R}}$, do the following
  – Reject if $(\mathsf{CRS}_1, \mathsf{CRS}_2)$ were not signed by S.
  – Run the extractor $\mathsf{Ext}_1(CRS_1, td_1, (pk, \hat{m}), \pi_1)$ to obtain a witness (m, r) for $(pk, \hat{m}) \in L_1$. Reject if $((pk, \hat{m}), (m, r)) \notin \mathcal{R}_{L_1}$.
  – Input m into $\mathcal{F}_{\mathsf{CONDEC}}$. Let s be the corresponding output.
  – Compute $(r', r'') \leftarrow H(pk, \hat{m}, \pi_1)$
  – Compute $\hat{s} \leftarrow \mathsf{FHE.Enc}_{pk}(s; r')$ and $\pi_2 \leftarrow \mathsf{Sim}_2(\mathsf{CRS}_2, td_2, (pk, c, \hat{m}, \hat{s}); r'')$.
  – Output $(\hat{s}, \pi_2)$ to $\mathcal{A}_{\mathsf{R}}$.

Let $\mathcal{Z}$ be a PPT environment. We will now prove indistinguishability between $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_{\mathsf{CONDEC}}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}_{\mathsf{CONDEC}}}(\mathcal{Z})$. Consider the following sequence of experiments.

Experiment 1 Simulator $\mathcal{S}_1$ simulates the real protocol Π_{CONDEC}.

Experiment 2 Identical to experiment 1, except that CRS_1 is computed by $(\mathsf{CRS}_1, td_1) \leftarrow \mathsf{FakeGen}_1(1^k)$.

Experiment 3 Identical to experiment 2, except that CRS_2 is computed by $(\mathsf{CRS}_2, td_2) \leftarrow \mathsf{FakeGen}_2(1^k)$.

Experiment 4 Identical to experiment 3, except that the token rejects if for the signature σ it holds that $\mathsf{SIG.Verify}_{vk}((\mathsf{CRS}_1, \mathsf{CRS}_2), \sigma) = 1$ even though $(\mathsf{CRS}_1, \mathsf{CRS}_2)$ were not signed by the S.

Experiment 5 Identical to experiment 4, except that the token now extracts m by $(m, r) \leftarrow \mathsf{Ext}_1(\mathsf{CRS}_1, td_1, (pk, \hat{m}), \pi_1)$ and rejects if $((pk, \hat{m}), (m, r)) \notin \mathcal{R}_{L_1}$ even though $\mathsf{V}_1(\mathsf{CRS}_1, (pk, \hat{m})) = 1$.

Experiment 6 Identical to experiment 5, except that the pseudorandom function F is replaced by a random oracle H.

Experiment 7 Identical to experiment 6, except that the proof π_2 is computed by $\pi_2 \leftarrow \mathsf{Sim}_2(\mathsf{CRS}_2, td_2, (pk, c, \hat{m}, \hat{s}); r'')$.

Experiment 8 Identical to experiment 7, except that the commitment c on a is replaced by a commitment c' on a random fake-input a'.

Experiment 9 Identical to experiment 8, except that the encrypted output $\hat{s}$ is computed by $\hat{s} \leftarrow \mathsf{FHE.Enc}_{pk}(\mathcal{F}_{\mathsf{CONDEC}}(a, m); r')$, where a is the sender-input. This is the ideal experiment.

Remarks The indistinguishability for experiment 1 and 2 as well as experiment 2 and 3 follows directly by the indistinguishability of real and fake common reference strings. Indistinguishability between experiment 3 and experiment 4 is established using the EUF-CMA-security of SIG. The indistinguishability of experiment 5 and experiment 6 easily follows by the pseudorandomness-property of $\{F_i\}$. The indistinguishability of experiment 7 and experiment 8 folows by the computational hiding property of the commitment-scheme com. Finally, the statistical indistinguishability of experiment 8 and experiment 9 follows by a simple hybrid argument using the circuit-privacy of FHE. We will now establish the remaining steps.

Lemma 16. *From $\mathcal{Z}$'s view, experiment 4 and experiment 5 are computationally indistinguishable, provided that $(\mathsf{CRSGen}_1, \mathsf{P}_1, \mathsf{V}_1)$ has the proof of knowledge property.*

Proof. Assume that $\mathcal{Z}$ distinguishes between experiment 4 and experiment 5 with non-negligible advantage $\epsilon(k)$. We will construct a corrupted prover P^* that defeats the knowledge-extractor Ext_1 of $(\mathsf{CRSGen}_1, \mathsf{P}_1, \mathsf{V}_1)$. Conditioned that $\mathcal{S}_4$ does not abort, experiment 4 and experiment 5 are identically distributed. Thus and environment $\mathcal{Z}$ distinguishing between experiment 4 and experiment 5 must provoke an abort of $\mathcal{S}_5$ with probability at least $\epsilon(k)$. Let $q(k)$ be a polynomial upper bound on the number of queries $\mathcal{A}_{\mathsf{R}}$ sends to $\mathcal{F}^{\mathsf{wrap}}_{\mathsf{stateless}}$. P^* first guesses an index $i \in [q(k)]$ of a query for which an abort is provoked. Moreover, P^* uses the fake common reference string it received as an input as CRS_1 rather than generating it with the $\mathsf{FakeGen}$ algorithm. P^* proceeds as $\mathcal{S}_4$, but once $\mathcal{A}_{\mathsf{R}}$ sends the i-th query $x_i = ((pk_i, \hat{m}_i), \pi_{1,i})$, P^* outputs $((pk_i, \hat{m}_i), \pi_{1,i})$. Clearly, from $\mathcal{Z}$'s views are identically distributed when interacting with $\mathcal{S}_4$ or P^*. Thus it holds that $\Pr[\mathsf{P}^* \text{ defeats } \mathsf{Ext}_1] = \Pr[\mathcal{S}_5 \text{ aborts with query } i] \geq \epsilon(k)/q(k)$ which is non-negligible. □

Lemma 17. *From $\mathcal{Z}$'s view, experiment 6 and experiment 7 are computationally indistinguishable, given that $(\mathsf{CRSGen}_2, \mathsf{P}_2, \mathsf{V}_2)$ is unbounded zero-knowledge.*

Proof. Assume that $\mathcal{Z}$ distinguishes experiment 6 and experiment 7 with non-negligible advantage $\epsilon(k)$. We will construct a distinguisher D that distinguishes between the Real and Ideal experiment for $(\mathsf{CRSGen}_2, \mathsf{P}_2, \mathsf{V}_2)$ with non-negligible probability. Instead of computing π_2 by invoking the

prover $\mathsf{P}_2(\mathsf{CRS}_2, (pk, c, \hat{m}, \hat{s}), (a, r, r'); r'')$ like $\mathcal{S}_6$, D forwards $((pk, c, \hat{m}, \hat{s}), (a, r, r')) \in \mathcal{R}_{L_2}$ to its oracle to obtain π. Otherwise $\mathcal{D}$ proceeds in the same way as $\mathcal{S}_6$. Clearly, if D is connected to the Real experiment, $\mathcal{Z}$'s view is distributed identically to experiment 6. Likewise, if D is connected to the Ideal experiment, then $\mathcal{Z}$'s view is distributed identically to experiment 7. Thus, D distinguishes between Real and Ideal with advantage $\epsilon(k)$. □

6.1.2 Corrupted Sender

We will next prove computational UC-security against a corrupted sender. Let $\mathcal{A}_\mathsf{S}$ be the corrupted sender, which is *w.l.o.g.* the dummy adversary. We first state the simulator $\mathcal{S}_\mathsf{S}$. Let $p(k)$ be a polynomial upper runtime bound for an honest receiver R. We need to assume such a runtime-bound for an honest receiver, since there is no efficient way to decide the bounded halting problem for a malicious token T^* other than running T^*. If the receiver R has an a-priori upper runtime-bound $p(k)$, the simulator can query the token T^* at least as often as an honest R would, thereby determining the number of invocations after which T^* halts.

Simulator $\mathcal{S}_\mathsf{S}$

- Set $(\mathsf{CRS}_1, td_1) \leftarrow \mathsf{FakeGen}_1(1^k)$ and $(\mathsf{CRS}_2, td_2) \leftarrow \mathsf{FakeGen}_2(1^k)$.
- Wait until $\mathcal{A}_\mathsf{S}$ sends a token T and a message (c, σ).
- Generate a pair of public and secret keys $(pk, sk) \leftarrow \mathsf{FHE.KeyGen}(1^k)$.
- Set $m \leftarrow 0^*$ to be the all-zero message.
- Set $\hat{m} \leftarrow \mathsf{FHE.Enc}_{pk}(m; r)$ for fresh coins r and $\pi_1 \leftarrow \mathsf{Sim}_1(td_1, (pk, \hat{m}))$
- Input $(pk, \hat{m}, \pi_1)$ into T. Let $(\hat{s}, \pi_2)$ be the corresponding output by T.
- If $\mathsf{V}_2(\mathsf{CRS}_2, (pk, c, \hat{m}, \hat{s}), \pi_2) = 0$ abort.
- Compute $(a, r', r'') \leftarrow \mathsf{Ext}_2(\mathsf{CRS}_2, td_2, (pk, c, \hat{m}, \hat{s}), \pi_2)$, abort if extraction fails.
- Repeat for $i = 1, \ldots, p(k)$:
  - Set $\hat{m}_i \leftarrow \mathsf{FHE.Enc}_{pk}(m)$ and $\pi_{1,i} \leftarrow \mathsf{Sim}_1(\mathsf{CRS}_1, td_1, (pk, \hat{m}_i))$.
  - Input $(pk, \hat{m}_i, \pi_{1,i})$ into T. Let $(\hat{s}_i, \pi_{2,i})$ be the corresponding output by T.
  - Abort loop if $\mathsf{V}_2(\mathsf{CRS}_2, (pk, c, \hat{m}_i, \hat{s}_i), \pi_{2,i})$ rejects.
- Let n_{invoc} be the number of the iterations after which the loop terminated.
- Input (a, n_{invoc}) into $\mathcal{F}_{\mathsf{CONDEC}}$.

Let $\mathcal{Z}$ be a PPT environment. We will now prove indistinguishability between $\mathsf{Real}^{\mathcal{A}_\mathsf{S}}_{\Pi_{\mathsf{CONDEC}}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_\mathsf{S}}_{\mathcal{F}_{\mathsf{CONDEC}}}(\mathcal{Z})$. Consider the following sequence of experiments.

Experiment 1 The simulator $\mathcal{S}_1$ simulates the real protocol Π_{CONDEC}.

Experiment 2 Identical to experiment 1, except that now CRS_1 is chosen by $(\mathsf{CRS}_1, td_1) \leftarrow \mathsf{FakeGen}_1(1^k)$.

Experiment 3 Identical to experiment 2, except that now CRS_2 is chosen by $(\mathsf{CRS}_2, td_2) \leftarrow \mathsf{FakeGen}_2(1^k)$.

Experiment 4 Identical to experiment 3, except that now $\mathcal{S}_4$ computes a by $(a, r', r'') \leftarrow \mathsf{Ext}_2(\mathsf{CRS}_2, td_2, (pk, c, \hat{m}, \hat{s}), \pi_2)$ and aborts if the extraction fails.

Experiment 5 Identical to experiment 4, except that now R's output is computed by $\mathcal{F}_{\mathsf{CONDEC}}(a, m)$.

Experiment 6 Identical to experiment 5, except that now the $\pi_{1,i}$ are computed by $\pi_{1,i} \leftarrow \mathsf{Sim}_1(\mathsf{CRS}_1, td_1, (pk, \hat{m}_i))$.

Experiment 7 Identical to experiment 6, except that now all messages $\hat{m}$ sent by R to T are replaced by all-zero messages 0^*. This is the ideal experiment.

Remarks The indistinguishability of experiments 1,2 and 3 again follows trivially from the indistinguishability of real and fake common reference strings. The indistinguishability of experiment 3 and experiment 4 follows by the proof of knowledge-property of the proof system $(\mathsf{CRSGen}_2, \mathsf{P}_2, \mathsf{V}_2)$. The indistinguishability of experiments 4 and 5 is established by the soundness-property of $(\mathsf{CRSGen}_2, \mathsf{P}_2, \mathsf{V}_2)$. The indistinguishability of experiment 5 and experiment 6 is established along the lines Lemma 17. Finally, the indistinguishability of experiment 6 and experiment 7 is established using the IND-CPA security of FHE, analogous to Lemma 15.

7 Common Random Reference Strings with Stateless Hardware

In this Section, we show how to emulate a common random reference string with a stateless hardware token. We will therefore UC-implement a coin-toss protocol between S and R. The result of that coin-toss can then be used as common reference string in protocol Π_{CONDEC}. Note that such a protocol must inherently make use of non-black-box techniques. It holds that any black-box simulation-strategy (that makes only black-box use of code of the token T) against a corrupted S leads to a successful adversarial strategy for a corrupted R. The reason for this is that any black-box use of the token by a sender-simulator, in particular rewinding, can be implemented by a corrupted receiver against a honest stateless token. Furthermore, such a coin-toss protocol requires interaction between S and R. The reason for this is that any simulation-strategy against a corrupted receiver can also be implemented by a stateful corrupted token T against an honest receiver. We will first outline the protocol Π_{CRS}. The idea is that S commits itself to some random coins $x \in \{0,1\}^k$ by hardwiring them into the token T. T will only unveil those coins once R knows the answer of a certain hard question. Specifically, R has to prove to T that it knows the preimage a of some value b under a one-way permutation F, i.e it knows an a such that $b = F(a)$. Once R sends his own random coins $y \in \{0,1\}^k$ to S, S sends x together with the answer a to R, computes $\mathsf{CRS} \leftarrow x \oplus y$ and outputs CRS. R can then prove knowledge of a to T and check whether T answers with the same y. If yes, R also computes $\mathsf{CRS} \leftarrow x \oplus y$ and outputs z. The critical part of this protocol is

the choice of the argument-system that is used to prove knowledge of the solution a to the token T. Since T is stateless, we have the requirement that the argument-system that is resettably sound.

We now provide a formal statement of protocol Π_{CRS} and a UC-security-proof. Let $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ be the stateless hardware wrapper-functionality, let $F : \{0,1\}^k \to \{0,1\}^k$ be a one-way permutation and (P, V) be a resettably sound zero-knowledge argument system of knowledge for the language $L = \{(k,b) \in \{0,1\}^k : \exists a \in \{0,1\}^k \text{ s.t. } b = F(a)\}$. Let $\ell = \mathsf{poly}(k)$ be the desired length of the output CRS.

Protocol Π_{CRS}

- Sender S:
  - Choose an $a \leftarrow_R \{0,1\}^k$ uniformly at random. Set $b \leftarrow F(a)$.
  - Choose an $x \leftarrow_R \{0,1\}^\ell$ uniformly at random.
  - Choose random private coins state_0 for V uniformly at random.
  - Program a stateless token T with the following functionality
    - Upon receiving a message $(\texttt{Verifier}, \tau)$, run $\mathsf{V}(\mathsf{state}_0, \tau)$ and output whatever V outputs.
    - Upon receiving a message $(\texttt{Unveil}, \tau)$, check if $\mathsf{V}(\mathsf{state}_0, \tau)$ accepts, if so output x, otherwise output $\perp$.
  - Input T into $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ and send b to R
- Receiver R:
  - Wait for the `ready` message from $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ and a message b from S.
  - Choose $y \leftarrow_R \{0,1\}^\ell$ uniformly at random.
  - Send y to S.
- Sender S:
  - Upon receiving a message y from R, send (x, a) to R and output $\mathsf{CRS} \leftarrow x \oplus y$
- Receiver R:
  - Wait for a message (x, a) from S.
  - Setup a prover P for the statement (k, b), witness-input a and fresh random coins r.
  - Simulate the computation of the prover P, by forwarding messages τ from P as input $(\texttt{Verifier}, \tau))$ to $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ and forwarding the output s of $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ to P. Stop the simulation once P terminates. Let τ the last message sent by P.
  - Input $(\texttt{Unveil}, \tau)$ into $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$. Let x' be the corresponding output.
  - Check if $x = x'$. If so, output $\mathsf{CRS} \leftarrow x \oplus y$, otherwise abort.

7.1 Proof of Security

We will now prove computational UC-security against both corrupted sender and receiver.

7.1.1 Corrupted Receiver

We will start with a corrupted receiver. Let $\mathcal{A}_{\mathsf{R}}$ be the dummy-adversary for a corrupted receiver. Let Ext be the knowledge-extractor for the argument of knowledge-system (P, V). We will now state the simulator $\mathcal{S}_R$.

Simulator $\mathcal{S}_{\mathsf{R}}$

- Simulate the first round of a sender S and forward the message b to $\mathcal{A}_{\mathsf{R}}$. Use the token-code T output by S to simulate the $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ functionality for $\mathcal{A}_{\mathsf{R}}$. Let a be the preimage of b under F.
- If $\mathcal{A}_{\mathsf{R}}$ sends a `Unveil`-query to $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$, reply $\perp$, regardless if V would accept.
- Let CRS be the output of the ideal $\mathcal{F}_{\mathsf{CRS}}$-functionality. Upon receiving a message y from $\mathcal{A}_{\mathsf{R}}$, set $x \leftarrow \mathsf{CRS} \oplus y$.
- Reprogram the token T to use the input x, send (x, a) to $\mathcal{A}_{\mathsf{R}}$ and proceed as in the real protocol.

Let $\mathcal{Z}$ be a PPT environment. We will now prove computational indistinguishability between $\mathsf{Real}_{\Pi_{\mathsf{CRS}}}^{\mathcal{A}_{\mathsf{R}}}(\mathcal{Z})$ and $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{CRS}}}^{\mathcal{S}_{\mathsf{R}}}(\mathcal{Z})$ for the case of a corrupted receiver. Consider the following sequence of experiments.

Experiment 1 Simulator $\mathcal{S}_1$ simulates the real protocol Π_{CRS}.

Experiment 2 Identical to experiment 1, except that $\mathcal{F}_{\mathsf{stateless}}^{\mathsf{wrap}}$ outputs $\perp$ if $\mathcal{A}_{\mathsf{R}}$ sends a `Unveil`-query for which the verifier V accepts before having sent his coins y to S.

Experiment 3 Identical to experiment 2, except that S's coins x are computed by $x \leftarrow \mathsf{CRS} \oplus y$, where CRS is an a-priori fairly chosen common random reference string. This is the ideal experiment.

Remarks Experiment 1 and experiment 2 are computationally indistinguishable, given that the permutation F is strongly one way. Experiment 2 and experiment 3 are identically distributed, as both x and CRS are uniformly and independently distributed.

Lemma 18. *From $\mathcal{Z}$'s view, experiment 1 and experiment 2 are computationally indistinguishable, given that F is a one-way permutation.*

Proof. From $\mathcal{Z}$'s view, experiment 1 and experiment 2 are identically distributed conditioned to the event that $\mathcal{A}_{\mathsf{R}}$ does not convince V before sending his own coins y. Thus, a $\mathcal{Z}$ distinguishing between experiment 1 and experiment 2 must convince V before sending y. Assume that $\mathcal{Z}$ causes this event with non-negligible probability $\epsilon(k)$. We will construct an adversary A that inverts the one-way permutation F with non-negligible probability. A simulates a prover P^*, by simulating the interaction of $\mathcal{Z}$ and $\mathcal{S}_1$. However, its set $b \leftarrow b^*$, where b^* is its own input, instead of computing b like $\mathcal{S}_1$. If $\mathcal{A}_{\mathsf{R}}$ sends its coins y to S, A aborts. If P^* convinces the simulated verifier V that it knows an a such that $b = F(a)$, A will run the extractor $\mathsf{Ext}((k, b), \mathsf{P}^*)$ to obtain a witness a. If it holds $F(a) = b^*$, A outputs a. Altogether, A inverts F with advantage $\epsilon(k)$. □

7.1.2 Corrupted Sender

Next, we will prove computational UC-security for the case of a corrupted sender. Let $\mathcal{A}_{\mathsf{S}}$ be the dummy-adversary for a corrupted sender and let Sim be the non-black-box simulator for the argument-system (P, V), that takes as input a statement (k, b) and the code V^* of a malicious verifier. The simulator $\mathcal{S}_S$ is given as follows.

Simulator $\mathcal{S}_{\mathsf{S}}$

- Let T^* be the stateful token sent by $\mathcal{A}_{\mathsf{S}}$ and b be the message sent by $\mathcal{A}_{\mathsf{S}}$.
- Construct from T^* a corrupted stateful verifier V^* that does the following. On input a message τ, simulate T^* on input $(\texttt{Verifier}, \tau)$ and keep its state.
- Run the non-black-box simulator Sim on the statement (k, b) and the verifier-code V^*. From the transcript created by Sim, take the messages $\tau_1, \ldots, \tau_n$ sent by the prover P.
- Input the messages $(\texttt{Verifier}, \tau_i)$ for $i = 1, \ldots, n$ into a new instance of T^*. Then input $(\texttt{Unveil}, \tau_n)$ to T^*. Let x be T^*'s output.
- Set $y \leftarrow \mathsf{CRS} \oplus x$, where CRS is the output of the ideal $\mathcal{F}_{\mathsf{CRS}}$-functionality.
- Send y to S. Let x' be the response of S.
- If $x \neq x'$ abort.

Let $\mathcal{Z}$ be a PPT environment. We will now prove computational indistinguishability between $\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_{\mathsf{CRS}}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}_{\mathsf{CRS}}}(\mathcal{Z})$ directly, given that the argument-system (P, V) is zero knowledge.

Lemma 19. *$\mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_{\mathsf{CRS}}}(\mathcal{Z})$ and $\mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}_{\mathsf{CRS}}}(\mathcal{Z})$ are computationally indistinguishable, provided that the argument-system (P, V) is computational zero-knowledge.*

Proof. Set $\mathbf{Exp}_1 = \mathsf{Real}^{\mathcal{A}_{\mathsf{R}}}_{\Pi_{\mathsf{CRS}}}(\mathcal{Z})$ and $\mathbf{Exp}_2 = \mathsf{Ideal}^{\mathcal{S}_{\mathsf{R}}}_{\mathcal{F}_{\mathsf{CRS}}}(\mathcal{Z})$. Denote $\mathbf{Exp}_i(r)$ when the environment $\mathcal{Z}$ is ran with coins r in $\mathbf{Exp}_i$. Assume for contradiction that $|\Pr[\mathbf{Exp}_1 = 1] - \Pr[\mathbf{Exp}_2 = 1]| \geq \epsilon(k)$ for a non-negligible $\epsilon(k)$. Then there exist coins r for the environment $\mathcal{Z}$ such that $|\Pr[\mathbf{Exp}_1(r) = 1] - \Pr[\mathbf{Exp}_2(r) = 1]| \geq \epsilon(k)$, otherwise the triangle inequality yields a contradiction to $|\Pr[\mathbf{Exp}_1 = 1] - \Pr[\mathbf{Exp}_2 = 1]| \geq \epsilon(k)$. Henceforth fix the coins of $\mathcal{Z}$ to be r. Let V^* be the verifier constructed in the second step of $\mathcal{S}_{\mathsf{S}}$ and (k, b) the statement sent by S. We can now construct a distinguisher D that distinguishes the distributions $\mathcal{D}_1 = \{\langle \mathsf{P}, \mathsf{V}^* \rangle(k, b)\}$ and $\mathcal{D}_2 = \{\mathsf{Sim}((k, b), \mathsf{V}^*)\}$ with advantage $\epsilon(k)$. Let X be a challenge-sample for D. D continues the simulation of Π_{CRS}, but takes the messages P sends to V from its sample X instead of running the prover P like in Π_{CRS}. D then continues the simulation and outputs whatever $\mathcal{Z}$ outputs. Clearly, if X is a sample from then $\mathcal{D}_1$, then D's output is identically distributed as $\mathbf{Exp}_1$. Likewise, if X is distributed according to $\mathcal{D}_2$, then D's output is distributed according to $\mathbf{Exp}_2$. Thus $|\Pr_{X \leftarrow_R \mathcal{D}_1}[D(X) = 1] - \Pr_{X \leftarrow_R \mathcal{D}_2}[D(X) = 1]| \geq \epsilon(k)$, contradicting the computational zero-knowledge property of (P, V). □

7.2 Using the same token for Π_{CRS} into Π_{CONDEC}

We conclude that protocol Π_{CRS} can be plugged into the protocol Π_{CONDEC} while using the same stateless hardware functionality. This can be done, as the only thing the token sees in this protocol is a zero-knowledge proof and we do not need to hide the common reference string from the token. This is the case because protocol Π_{CONDEC} requires the token to learn the common reference strings in order to verify and proof statements. However, we chose to state the two protocols separately for ease of presentation.

8 Conclusion

In this work, we constructed a reusable general purpose obfuscation-scheme based on a single stateless hardware token, that prohibits adaptive behavior by a malicious token. Since the VBB-obfuscation of the conditional decryption functionality is impossible (as follows readily from our first result), the use of a single stateless token seems to be a minimal setup assumption. Our scheme is even secure against an unbounded sender, as long as the token's computational power is restricted. An interesting application of the latter feature is serialization of efficient two-party computations, where one party is computationally unbounded. The idea is that the computationally unbounded party sends her encrypted inputs together with the obfuscated program to the computationally bounded party, who executes the obfuscated program.

References

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[Bar01] B. Barak. How to go beyond the black-box simulation barrier. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:106, 2001.

[BCCT11] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. Cryptology ePrint Archive, Report 2011/443, 2011. `http://eprint.iacr.org/`.

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS*, pages 116–125, 2001.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:109, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO*, pages 455–469, 1997.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. Obfuscating point functions with multibit output. In *EUROCRYPT*, pages 489–508, 2008.

[CKS+11] Seung Geol Choi, Jonathan Katz, Dominique Schrder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (effient) universally composable two-party computation using a minimal number of stateless tokens. personal communication, 2011.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[DFH11] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. Cryptology ePrint Archive, Report 2011/508, 2011. `http://eprint.iacr.org/`.

[DKMQ11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *TCC*, pages 164–181, 2011.

[DN07] Cynthia Dwork and Moni Naor. Zaps and their applications. *SIAM J. Comput.*, 36(6):1513–1543, 2007.

[FLS90] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, SFCS '90, pages 308–317 vol.1, Washington, DC, USA, 1990. IEEE Computer Society.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Proceedings of Eurocrypt 2006, volume 4004 of LNCS*, pages 339–358. Springer, 2006.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC*, pages 194–213, 2007.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In *EUROCRYPT*, pages 54–71, 2009.

[Had10] Satoshi Hada. Secure obfuscation for encrypted signatures. In *EUROCRYPT*, pages 92–112, 2010.

[HRSV07] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *TCC*, pages 233–252, 2007.

[LPS04] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, pages 20–39, 2004.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, pages 33–43, 1989.

[NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC '90, pages 427–437, New York, NY, USA, 1990. ACM.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 433–444, London, UK, 1992. Springer-Verlag.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. *SIAM Journal on Computing*, 20:1084–1118, 2001.

[SG11] Aviad Rubinstein Shafi Goldwasser, Huijia Lin. Delegation of computation without rejection problem from designated verifier cs-proofs. Cryptology ePrint Archive, Report 2011/456, 2011. `http://eprint.iacr.org/`.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, pages 523–532, 2005.