On Efficient Non-Interactive Oblivious Transfer with Tamper-Proof Hardware

Maria Dubovitskaya* Alessandra Scafuro† Ivan Visconti‡

Abstract

Oblivious transfer (OT, for short) [Rab81] is a fundamental primitive in the foundations of Cryptography. While in the standard model OT constructions rely on public-key cryptography, only very recently Kolesnikov in [Kol10] showed a truly efficient string OT protocol by using tamper-proof hardware tokens. His construction only needs few evaluations of a block cipher and requires stateless (therefore resettable) tokens that is very efficient for practical applications. However, the protocol needs to be interactive, that can be an hassle for many client-server setting and the security against malicious sender is achieved in a *covert* sense, meaning that a malicious sender can actually obtain the private input of the receiver while the receiver can detect this malicious behavior with probability 1/2. Furthermore the protocol does not enjoy forward security (by breaking a token one violates the security of all previously played OTs).

In this work, we propose new techniques to achieve efficient *non-interactive* string OT using tamper-proof hardware tokens. While from one side our tokens need to be stateful, our protocol enjoys several appealing features: 1) it is secure against malicious receivers and the input privacy of honest receivers is guaranteed unconditionally against malicious senders, 2) it is forward secure, 3) it enjoys adaptive input security, therefore tokens can be sent before parties know their private inputs. This gracefully fits a large number of client-server settings (digital TV, e-banking) and thus many practical applications. On the bad side, the output privacy of honest receivers is not satisfied when tokens are reused for more than one execution.

Keywords: OT, Tamper-Proof Hardware Tokens.

1 Introduction

Oblivious transfer (OT) [Rab81, EGL85] is among the most investigated primitives in modern Cryptography. Its importance is due to several reasons. OT is conceptually useful on his own, and as subprotocol, for realizing secure multi-party computation [IPS08, Kil88] and in general for cryptographic protocol design. Unfortunately in the standard model OT constructions rely on public-key encryption, which limits its practical applicability when OT is used as subprotocol and several instances of it have to be executed by the larger protocol. This efficiency problem is even more critical when one would like to use lightweight cryptographic hardware as smart cards or RFID chips.

Among different assumptions proposed in the past to overcome the impossibility results in the standard model, the use of tamper proof hardware tokens has obtained an increasing interest for its capability of designing elegant and extremely powerful protocols. First of all, tamper-resistant hardware tokens implementing some well-known cryptographic tasks are widely available (e.g.,

*IBM Research - Zurich, SWITZERLAND. E-mail: mdu@zurich.ibm.com.

†Dip. di Informatica ed Appl., University of Salerno, ITALY. E-mail: scafuro@dia.unisa.it.

‡Dip. di Informatica ed Appl., University of Salerno, ITALY. E-mail: visconti@dia.unisa.it.

smart cards) and have been used in practice in the last decades. Second, it has been shown that protocols benefit from the use of such tokens either in practice obtaining better computational and communication complexities, and/or from a theoretic point of view, obtaining feasibility results otherwise impossible to achieve.

Hardware tokens are supposed to carry out some publicly known tasks such that any player can initialize them with their own secrets. Then tokens are distributed to the other players (honest or malicious) that can use them in a "black box" way, i.e., by sending an input to the token and getting back only the output without obtaining any information about its internal state. Obviously nothing can prevent a malicious player from constructing adversarially designed tokens on its own, and thus it is challenging to design advanced cryptographic protocols that benefit from the use of honest tokens, and simultaneously maintain all the desired security guarantees in presence of adversarially designed tokens.

1.1 Evaluating Protocols with Hardware Tokens

There are several physical properties that could be enjoyed by tokens used in a cryptographic protocol. Subtle differences can heavily simplify or complicate the design of a cryptographic protocol. Before discussing related work, here we take a chance to summarize and discuss requirements, and features of tamper-proof hardware tokens proposed in literature, and their impact on the efficiency and security of the protocols. This will simplify our discussion on previous work and will clearly place our contribution in the state-of-the-art.

Token requirements. We list below specific features/assumptions that differentiate heterogeneous tokens.

Resettability: a resettable token can be reset to a previous state, i.e., the adversary is able to manipulate the token and reset its internal variables (e.g., a counter) to its initial state. This is a positive property for a protocol, as it decreases the hardware assumption (i.e., the protocol does not need the additional hardware assumption that a token can not be reset).

State assumption: it is sometimes the case that next computations of a token are based on previous ones, in particular depending on the content of the answered queries. Indeed, when answering queries, a token could for instance erase some data, increment a counter and so on. Such tokens have to reliably keep their state even without the power supply, and are referred to as *stateful* in contrast to *stateless* tokens, that instead do not require any permanent updatable memory. Obviously a stateless token require a less demanding hardware assumption than a stateful token.

Knowledge of the code: sometimes security proofs rely on the ability of the simulator to rewind the tokens received from potentially malicious parties. This means that the adversary that produced the token is supposed to "know" its code. Such an assumption prevents to model real-life adversaries who may simply pass on hardware tokens obtained from one party to another party, or may construct a *super*-token by encapsulating received tokens, without actually knowing its content. This assumption is therefore problematic to justify when one wants to use such tokens to prove results under concurrent compositions. Indeed in these scenarios, it is natural to consider a man-in-the-middle adversary that passes on tokens, or that builds super tokens on top of some other tokens.

Quantity and directions: the introduction of physical tokens in a distributed computation becomes hard to justify when the number of tokens required to carry out the computation is large, and tokens have to be generated by many parties. Indeed, given the problematic

aspects of producing and exchanging tokens one would like to see only a small number of tokens circulating in the system, possibly in one direction only, therefore matching several client-server scenarios (e.g., often users after a subscription to a service receive a smart card).

Re-usability and adaptive inputs: given the overhead of exchanging tokens and in some cases, their manufacturing cost, an important property for the design of practical protocols is that tokens should be reusable any polynomial number of times. Moreover when the token is sent, the inputs for the computations that will be later performed should not necessarily be known yet and can be dynamically decided according to the outputs of the already performed computations.

Efficiency. Here we stress some more standard measures.

Number of rounds represents the number of communication rounds played by parties. This is a classical measure in cryptographic protocols, and non-interactiveness makes a protocol suitable for a much larger number of applications.

Number of queries represents the number of times a token is required to answer a query. For practical reasons tokens are usually lightweight devices with a slow communication interface, therefore minimizing the number of queries is worthy.

Amount of computations represents the amount of computations required both by parties and tokens. This is also a classical measure with a specific focus here because of the power constraints of available tamper-proof tokens. In this context even the use of symmetric key encryption in place of public key encryption turns out to be a big improvement in efficiency. In particular among the required computations, one would like to minimize the amount of computations done by each token in each execution of the protocol.

Security issues. We now point out some security issues when dealing with hardware tokens.

Protocol composition: an important measure for the security of a protocol is whether it maintains the security guarantees also when played in composition with itself, or with other protocols. Therefore one has to argue whether a protocol enjoys one-time, sequential, self-concurrent, general concurrent or universal composition.

Adversary's power: an adversary can be semi-honest (i.e., a protocol guarantees input/output privacy as long as the adversary follows honestly the protocol and then tries to analyze its view), covert (i.e., a protocol can leak some input/output privacy of honest players, as long as there is a reasonable chance to catch the malicious party) and malicious (i.e., a protocol must protect honest players' input/output privacy from any adversarial behavior). Moreover, security can be based on computational assumptions, therefore against efficient adversaries only, or unconditional, therefore against unbounded adversaries.

Cryptographic assumptions and models: the use of general assumptions should be preferred since a protocol can then be instantiated under various candidate assumptions. Moreover, the use of random oracles (which limits the adversary to a black-box use of a collision-resistant hash function) and in general of controversial conjectures should be avoided.

Forward security: in case the adversary at some point breaks the tamper resistance of the token and obtains its current state, previous computations should remain secure.

1.2 Related Work

The use of tamper-proof hardware tokens for cryptographic purposes was investigated by Goldreich and Ostrovsky focusing on oblivious RAM [GO96]. In [GKR08] Goldwasser, Kalai and Rothblum introduced the new paradigm of one-time computing achieved by means one-time programs: a (semi-honest) sender embeds a program in a token that is sent to a malicious receiver. The receiver executes the program embedded in the token only once. The use of tamper-proof hardware that is sent by a party P_1 to a party P_2 and that cannot interact back to the party P_1 can be seen as a an application of the two-prover model of [BGKW88] where it is showed how two provers that cannot communicate with each other allow one to achieve unconditional zero knowledge protocols. Moran and Naor [MN05] considered a relaxation of tamper-proof hardware called "tamper-evident seals", and demonstrated the possibility of implementing cryptographic primitives based on this relaxed notion. Then Katz in [Kat07] put forth a more general formalization of a tamper-proof token model. He provides a scheme achieving Universally Composable (UC-secure) two-party computation under the DDH assumption. His construction uses tokens assumed to be equipped with a built-in source of randomness (this can be more concretely implemented by using stateful tokens and pseudo-random functions) that cannot be reset and requires that tokens be sent by both parties. In [Kat07], it is assumed that once a party creates a hardware token and sends it off, it can not send any messages to the token (but can receive messages from it). Also it is assumed that all parties (including the malicious ones) know the code run by the hardware token that they distributed, which technically means that tokens can be rewound.

Chandran, Goyal and Sahai [CGS08] extended the results of Katz and suggested three main improvements: first, they considered resettable tokens, second, their construction is based on general assumption (enhanced trapdoor permutations) instead of DDH. And finally, their security proof does not rely on the simulator's ability to rewind hardware tokens, thus they make no assumptions on how malicious parties create the hardware token which they distribute. As for communication between a token and its creator they require the opposite assumption: a token can not send any messages to its creator, but can potentially receive messages from it.

Moran and Segev [MS08] realized the UC commitment functionality in the same model of [Kat07] but providing several improvements. In their constructions tokens are passed in one direction only, therefore fitting all real-life scenarios which are inherently asymmetric (e.g, voting systems). Furthermore they showed a protocol realizing the commitment functionality without relying on computational assumptions and proved that the existence of one-way functions suffices to realize the multiple commitment functionality.

We notice that both the results of [CGS08, MS08] focus on constructing UC commitments. Then the compiler of [CLOS02] must be used to obtain security for general functionalities. Therefore it turns out that both constructions do not achieve efficiently the oblivious transfer functionality.

Goyal, Ishai, Sahai and Wadia in [GIS$^+$10] considered the general question of basing cryptography on tamper-proof tokens, under minimal computational assumptions. They considered both stateful and stateless tokens and, in particular, they showed that by exchanging a polynomial number of simple stateful hardware tokens, any functionality can be realized with unconditional security against malicious parties without interaction, and that stateless tokens and one-way functions are sufficient for UC-secure computation. In the stateless construction they require a polynomial number of rounds. These constructions however constitute mainly feasibility results since requiring either a polynomial number of tokens or a polynomial number of rounds is not practical for real life applications.

Very recently Döttling, Kraschewski and Müller-Quade in [DKMQ11] provided a protocol

achieving unconditional universal composable Oblivious Transfer using a single token. The protocol is interactive and interestingly provides bounded reusability (i.e., the same token can be re-used for a bounded number of executions).

Goyal, Ishai, Mahmoody and Sahai in [GIMS10] focusing on feasibility results addressed the challenging issue of achieving unconditional security by using only stateless token. On top of the model proposed in [CGS08] where parties do not need to know the token's code, they prove that if token encapsulation is possible (i.e., a token can be used to encapsulate other tokens) then unconditional UC-secure OT is possible. Furthermore, sticking on feasibility results they prove that if encapsulation is not possible then there exist no protocols using stateless tokens and achieving statistically secure OT.

The work of [HL08] provides an efficient protocol for secure set intersection and extensions to Oblivious Database Search with a stateful trusted token. The security proof crucially relies on the fact that the token is trusted that helps significantly the simulation. Furthermore the token can be used only for one protocol execution thus reusability is not provided. The question of using simple tamper-proof hardware tokens that are also reusable and suite for practical secure computation has been addressed recently by Kolesnikov [Kol10]. On top of some new techniques and a careful use of strong PRPGs, Kolesnikov presented an efficient protocol for string OT, relying on resettable (and actually, stateless) tamper-proof tokens. This protocol is secure against covert sender and malicious receiver under sequential composition. All parties, including the tokens only have to run few evaluations of a block cipher. Security against covert sender means that the input/output privacy of the receiver can be compromised, but there is a deterrence factor that can expose the cheating behavior of the sender, and this can be sufficient in some applications. If the token is semi-honest (e.g., if it is provided by a trusted entity, but adversarially initialized), then his protocol is secure against malicious adversaries under concurrent general composition. However this last model assumes that an adversary can not create adversarially designed tokens, which is unrealistic in practice. Improvements of the work of [HL08] have been recently shown in [FPS+11].

1.3 Our Contribution

We propose a non-interactive string oblivious transfer (OT) protocol based on stateful tamper-proof tokens and the existence of one-way functions. From a practical point of view, the only computations required by each token consist of few evaluations of forward-secure PRGs. We obtain the optimal round complexity by means of a careful use of two *stateful* hardware tokens. These tokens are created by one party (the sender) and then distributed to the other party (the receiver). This particularly fits the standard client-server setting where a sender provides the users of all the inputs and the user is supposed not to send any message back to the server but only to decrypt the messages received (digital TV).

In our protocol the receiver will play separately with the received tokens that are not able to communicate directly with each other (the same idea as the two prover model of [BGKW88]). Jumping ahead, we will show that one can move the interaction from the communication network that connects remote parties to the location of one party only that can run a protocol with each token. We stress that tokens are sent from the sender at the very beginning, when parties do not have to know their inputs yet. We achieve a more general definition of OT (adaptive-input OT [Lin04]) that allows parties to choose their inputs adaptively. This implies that tokens can be sent before the parties choice their inputs and the inputs can be decided adaptively upon each execution.

In our protocol the communication between parties is done only in one direction (from sender to receiver), so the protocol is *non-interactive*. After the receiver gets tokens from the sender it

queries each token once and obtains the selected string. Neither a token nor the sender know about the receiver's choice.

Our protocol guarantees unconditional input privacy against malicious senders. The output privacy of a honest receiver is guaranteed only if tokens are not reused. When token are re-used the receiver could obtain an output dependent on previous inputs. This attack cannot be simulated, on the other hand it does not convey any information about the receiver's input neither to the sender nor to the tokens. Thus, in the following, when referring to the security property that the protocol enjoys we assume that with respect to a malicious sender only input privacy is preserved. We stress that when using such a security notion in a larger protocol, the fact that output privacy does not hold could also compromise input privacy.

We prove our protocol secure under sequential composition. Furthermore, our protocol is forward secure, i.e., if the adversary at some point breaks the token and obtains its current state, all future computations will be insecure, but the previous computations of the sender still remain secure.

Our protocol also provides token re-usability, which means that once tokens are sent to the receiver they can be used to perform polynomial many protocol executions, although the output privacy of the honest receiver can be compromised.

We assume that tokens can be rewound, i.e., the sender needs to know the content of the tokens that he sends.

In terms of efficiency, our protocol can be compared to Kolesnikov's protocol. Our proposal is computationally slightly more efficient, and moreover non-interactive. Although compared to [Kol10], we use 2 stateful tokens, we prove that in our protocol input privacy is forward secure against malicious receivers. The results of [Kol10] do not enjoy forward security and moreover either have weaker security guarantees (i.e., security is proved in the covert model of [AL07]) for both input and output privacy, or rely on a semi-honest token.

In our case the input privacy of the receiver is preserved unconditionally (due to the non-interactiveness) while the output privacy is unconditional only when tokens are used once.

In our security proof the simulator as in [Kat07, MS08, GIS$^+$10] rewinds the token, which means that the sender needs to know the code executed by the token that it sent. This implies that an adversary never forwards to a party a token received from another party, and moreover it never constructs a new token on top of a received one. Given the above evident limitations in a truly concurrent setting, we therefore prove the protocol secure under sequential composition, with respect to input privacy only. Instead in [Kat07, MS08, GIS$^+$10] security under general concurrent composition has been proved still under the above limitation about forwarding or extending some received tokens. The construction given in [CGS08] instead does not suffer of this limitation and gives a proof of UC security without further weaknesses in the adversary model. Nevertheless this is obtained by loosing other important properties such efficiency in terms of round and computation complexity.

2 Definitions and Tools

Notation. We denote by n the security parameter and by PPT the property of an algorithm of running in probabilistic polynomial-time. A function $\mathsf{negl}(\cdot)$ is negligible in n (or just negligible) if for every polynomial $p(\cdot)$ there exists a value N such that for all $\mathsf{n} > N$ it holds that $\mathsf{negl}(n) < 1/p(n)$.

Let $X = \{X(n,a)\}_{n \in N a \in \{0,1\}^*}$ and $Y = \{Y(n,a)\}_{n \in N, a \in \{0,1\}^*}$ be distribution ensembles. We say that X and Y are computationally indistinguishable, (i.e., $X \stackrel{c}{\equiv} Y$), if for every non-uniform PPT distinguisher D there exists a negligible function $\mathsf{negl}(\cdot)$, such that for every $a \in \{0,1\}^*$,

$$\Big|Pr[D(X(n,a))=1] - Pr[D(Y(n,a))=1]\Big| < \mathsf{negl}(n).$$

We denote by $y \xleftarrow{\$} \mathsf{B}(x)$ the value assigned to variable y as the output of the PPT algorithm B on input x, while with $y \leftarrow \mathsf{B}(x)$ we indicate the output of the deterministic polynomial-time algorithm B. For a finite set A, $x \xleftarrow{\$} \mathsf{A}$ denotes the assignment of a uniformly chosen element of A to variable x.

Pseudo-random generators. A pseudo-random generator (PRG) $\mathcal{G}{:}\{0,1\}^{\mathsf{n}} \rightarrow \{0,1\}^{\mathsf{n+k}}$ is a deterministic polynomial-time algorithm that receives a short truly random seed and stretches it into a longer string (expansion) that is indistinguishable from a truly random string (pseudo-randomness).

Here we consider a more powerful adversary that at some point is able to break into the computing device and thus to obtain its current state. While in this case all future computations will obviously be completely insecure, here we require that the security (pseudo-randomness of the generated bits) of previous computations still holds. This is a stronger security notion that is referred to as forward-security and requires that the previous output of the PRG remains secure even after the adversary gets the current state. Bellare and Yee in [BY03] provide a comprehensive treatment of forward-security in the context of shared-key based cryptographic primitives investigating on definitions and constructions of forward secure PRGs. We follow their definition of forward secure pseudo-random generator that we recall below.

Forward secure pseudo-random generators. A quadruple of efficient algorithms $\mathsf{GEN} = (\mathsf{GEN.key}, \mathsf{GEN.next}, \mathsf{k}, \mathsf{t})$ is a stateful generator, when k and t are two positive integers indicating respectively the stretch factor and the maximum number of blocks the generator can produce, $\mathsf{GEN.key}$ is a probabilistic key generation algorithm that outputs the initial state (the seed) and $\mathsf{GEN.next}$ is the (deterministic) next step algorithm that on input the current state returns a pair consisting of a k-bit string output block and the next state. A sequence $Out_1, Out_2, \ldots, Out_{\mathsf{t}}$ of k-bit output blocks is obtained by picking a seed $St_0 \xleftarrow{\$} \mathsf{GEN.key}$ and then iterating $(Out_i, St_i) \leftarrow \mathsf{GEN.next}(St_{i-1})$ for $i = 1, \ldots, \mathsf{t}$. St_{i-1} can be seen as the "key" or "seed" at time i. Forward security will require that this key is erased as soon as the next one has been generated, so that someone breaking into the machine gets only the current key.

Forward security. The forward security property is modeled by the following security experiment. The adversary $\mathcal{A}$ runs in two stages: in the "find" stage it receives output blocks, one at a time, until it (adaptively) decides to break in and obtains the current state. In the "guess" stage, it must decide whether the output blocks it had been fed were outputs of the generator or were independent random bits. Let $\mathcal{A}$ (find, Out, h) denote $\mathcal{A}$ in the find stage, taking an output block Out and current history h and returning a pair (d, h) where h is an updated history and $d \in \{\mathsf{find}, \mathsf{guess}\}$. This stage continues until $d = \mathsf{guess}$ or all t output blocks have been generated (in the latter case the adversary is given the final state in the guess stage). More formally, the adversary will play in one of the following experiments:

Experiment $\mathbf{Exp}^{\text{fprg-1}}_{\mathsf{GEN}}(\mathcal{A})$	Experiment $\mathbf{Exp}^{\text{fprg-0}}_{\mathsf{GEN}}(\mathcal{A})$
$St_0 \xleftarrow{\$} \mathsf{GEN.key}$	$St_0 \xleftarrow{\$} \mathsf{GEN.key}$
$i \leftarrow 0; h \leftarrow \epsilon$	$i \leftarrow 0; h \leftarrow \epsilon$
`Repeat`	`Repeat`
$i \leftarrow i + 1$	$i \leftarrow i + 1$
$(Out_i, St_i) \leftarrow \mathsf{GEN.next}(\mathsf{St_{i-1}})$	$(Out_i, St_i) \leftarrow \mathsf{GEN.next}(\mathsf{St_{i-1}})$
	$Out_i \xleftarrow{\$} \{0,1\}^{\mathsf{k}}$
$(d, h) \xleftarrow{\$} A(\mathsf{find}, Out_i, h)$	$(d, h) \xleftarrow{\$} A(\mathsf{find}, Out_i, h)$
`Until` $(d = \mathsf{guess})$ or $(i = \mathsf{t})$	`Until` $(d = \mathsf{guess})$ or $(i = \mathsf{t})$
$g \xleftarrow{\$} A(\mathsf{guess}, St_i, h)$	$g \xleftarrow{\$} A(\mathsf{guess}, St_i, h)$
`Return` g	`Return` g

Definition 1 (Forward Secure Pseudo-Random Generator)**.** *Let* GEN *be a stateful generator.* GEN *is a forward secure pseudo-random generator if for all non-uniform PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\mathbf{Adv}^{\text{fprg}}_{\mathsf{GEN}}(\mathcal{A}) = \left| Pr[\mathbf{Exp}^{\text{fprg-0}}_{\mathsf{GEN}}(\mathcal{A}) = 1] - Pr[\mathbf{Exp}^{\text{fprg-1}}_{\mathsf{GEN}}(\mathcal{A}) = 1] \right| < \mathsf{negl}(\mathsf{n})$.

Symmetric-key encryption. A symmetric encryption scheme ES is a pair of efficient algorithms $(\mathsf{Enc}, \mathsf{Dec})$ where Enc is used for data encryption, and Dec is used for data decryption. More precisely, the encryption algorithm Enc takes as input a randomly selected private key K from a key space, and a plaintext m from a message space, and outputs a ciphertext c belonging to a ciphertext space. The decryption function takes as input a randomly selected private key K from a key space and a ciphertext c belonging to a ciphertext space, and outputs a message m belonging to a plaintext space. The correctness requirement of an encryption scheme is that for any key K selected from the key space and any message m selected from the message space, it holds that $\mathsf{Dec}(K, \mathsf{Enc}(K, m)) = m$.

CPA-secure encryption. Consider the following security experiment: an IND-CPA challenger, selects an encryption key $K \leftarrow \{0,1\}^{\mathsf{n}}$ (here we assume wlog that the key space consists of all n-bit strings). The adversary can challenge the encryption oracle $\mathcal{O}$ giving as input plaintexts and receiving back the corresponding ciphertexts computed using K. Then the adversary sends two messages m_0 and m_1 belonging to the plaintext space to the IND-CPA challenger. The IND-CPA challenger randomly chooses a bit $b \in_R \{0,1\}$, computes $c_b \leftarrow \mathsf{Enc}(K, m_b)$, and returns the resulting ciphertext c_b to the adversary. The adversary can again challenge the encryption oracle $\mathcal{O}$. Finally the adversary must return a bit b' and wins if c_b encrypts m'_b (i.e., if $b = b'$). A symmetric-key encryption scheme is secure against *chosen plaintext attacks* if every non-uniform PPT adversary wins the above game with negligible advantage over $1/2$.

2.1 1-out-of-2 Oblivious Transfer

1-out-of-2 string Oblivious Transfer (OT) is a two-party protocol between a sender S and a receiver R. The sender S has two secret strings s_0, s_1, and the receiver R has a selection bit i from $\{0,1\}$. Upon completion, R learns s_i, but nothing about $s_{(1-i)}$, and S learns nothing about i. Following the notation of [Lin04] the Oblivious Transfer functionality is a function: $\mathcal{F}_{OT} : ((s_0, s_1), i) \mapsto (\varepsilon, s_i)$ where the sender gets no output. In this work we consider the case in which parties run sequentially a polynomial number of OT protocol executions and we require that at each execution players are allowed to select their inputs adaptively on outputs obtained

in all previous executions. We refer to this variant of OT as $\mathcal{F}_{adpt}^{OT}$ and we follow the definition of functionality with adaptively chosen inputs adopted by Lindell in [Lin04].

Sequential self-composition and stateful parties. The notation used in the above $\mathcal{F}_{adpt}^{OT}$ allows through session ids the execution of multiple OT instances. When multiple instances are possible, a natural question is whether the adversary can interleave the execution of different instances or not. Here we will consider an adversary that mounts a sequential attack, therefore multiple instances of an OT protocol will not be interleaved, and thus they will be played sequentially. The adversary will use his control of the communication network. Among his features, it will be able to ask the honest sender to play OT sessions concerning different senders, as also the same stateful senders. The only restriction concerns the fact that before a new OT starts, all the messages of the previous execution must have been exchanged, or otherwise the previous execution will never be completed.

Static corruption. Concerning the corruption capabilities of the adversary, we will consider a static adversary only, which means that the adversary selects the party to be impersonated when the experiment starts. This means that in all executions, the adversary will always play as sender or will always play as receiver.

Adaptively chosen inputs. When multiple executions of a protocol are possible, each party needs an input for each execution. It is often the case that a party chooses the input of each execution adaptively basing its decision on the outputs of previous executions. Following the definition of Lindell in [Lin04] this property is formalized by requiring that inputs of each execution be provided by a PPT input-selecting Touring Machine M. More formally, each honest party P_i is augmented by a PPT input-selecting Touring Machine M_i and the input played in the session sid is the result of the computation of $M_i(\mathsf{init}_{P_i}, \mathsf{sid}, \alpha_{\mathsf{sid},1}, \ldots, \alpha_{\mathsf{sid},j})$ where init_{P_i} is the initial state of P_i, sid is the session id of the current session, and $\alpha_{\mathsf{sid},1}, \ldots, \alpha_{\mathsf{sid},j}$ are outputs obtained from the sessions concluded so far. Note that init contains the initial inputs along with some possible auxiliary information.

Ideal model execution. The ideal functionality is shown in Fig. 1. We let Sim be the non-uniform PPT ideal world adversary. Then, the ideal execution of $\mathcal{F}_{adpt}^{OT}$ (with security parameter n, input-selecting machines $\overline{M} = (M_1, M_2)$, initial inputs $(\mathsf{init}_\mathsf{S}, \mathsf{init}_\mathsf{R})$ and auxiliary input z to Sim), denoted $\mathbf{Ideal}_{\mathcal{F}_{adpt}^{OT},\mathsf{Sim},\overline{M}}(\mathsf{n}, \mathsf{init}_\mathsf{S}, \mathsf{init}_\mathsf{R}, z)$ is the random variable defined as the output pair of the honest party and Sim from the above ideal execution.

Real model execution. In the real model execution the $\mathcal{F}_{adpt}^{OT}$ functionality is computed by a protocol π in which S and R are defined as two sets of instructions π_S and π_R respectively, that are computable in polynomial time. We let $\mathcal{A}$ be a non-uniform PPT adversary that controls either S or R and the scheduling of all messages throughout the (sequential) executions of the sessions. The (sequential) execution of π (with security parameter n, input-selecting machines $\overline{M} = (M_1, M_2)$, initial inputs $(\mathsf{init}_\mathsf{S}, \mathsf{init}_\mathsf{R})$, and auxiliary input z to $\mathcal{A}$), denoted by $\mathbf{Real}_{\pi,\mathcal{A},\overline{M}}(\mathsf{n}, \mathsf{init}_\mathsf{S}, \mathsf{init}_\mathsf{R}, z)$, is the random variable defined as the output pair of the honest party and $\mathcal{A}$ resulting from the following process.

The session sid is initiated by the adversary by sending a *start-session* message to the honest party. The honest party then applies its input-selecting machine on its initial input, the session number sid and its previously received outputs, and obtains the input for this new session. Upon

Functionality $\mathcal{F}_{adpt}^{OT}$

$\mathcal{F}_{adpt}^{OT}$ proceeds as follows, running with an oblivious transfer sender S, a receiver R and an adversary Sim that controls one of the parties:

1. Upon receiving a message (`start session`) from Sim send (`start session`, sid) to the honest party P_i where sid is the index of the session. The honest party P_i then applies its input-selecting machine M_i to its initial input init_{P_i}, the session number sid and the previous outputs and obtains the input x_{sid} (i.e., $(s_0^{\mathsf{sid}}, s_1^{\mathsf{sid}})$ for S, i_{sid} for R) that is $x_{\mathsf{sid}} \stackrel{\$}{\leftarrow} M_i(\mathsf{init}_{P_i}, \mathsf{sid}, \alpha_{\mathsf{sid},1}, \ldots, \alpha_{\mathsf{sid},j})$. Finally P_i sends $(\mathtt{role}, \mathsf{sid}, x_{\mathsf{sid}})$ to the ideal functionality (with $\mathtt{role} \in \{\mathtt{sender}, \mathtt{receiver}\}$).
2. Upon receiving a message $(\mathtt{role}, \mathsf{sid}, x_{\mathsf{sid}})$ from an honest party P_i, if no other message matching $(\mathtt{role}, \mathsf{sid}, \cdot)$ has been already received from P_i, then store $(\mathtt{role}, \mathsf{sid}, x_{\mathsf{sid}})$ otherwise discard it.
3. Upon receiving a message $(\mathtt{receiver}, \mathsf{sid}, i)$ from Sim, if a message matching $(\mathtt{sender}, \mathsf{sid}, (s_0, s_1))$ has been previously stored, then send (sid, s_i) to Sim, otherwise discard $(\mathtt{receiver}, \mathsf{sid}, i)$.
4. Upon receiving a message $(\mathtt{sender}, \mathsf{sid}, (s_0, s_1))$ from Sim, if no other message matching $(\mathtt{sender}, \mathsf{sid}, (\cdot, \cdot))$ has been already received from Sim then store $(\mathtt{sender}, \mathsf{sid}, (s_0, s_1))$ otherwise discard it.
5. Upon receiving a message $(\mathtt{send\text{-}output}, \mathsf{sid})$ from Sim send the output (sid, s_i) to the honest party. (If (sid, s_0, s_1) and (sid, i) have not yet been received by the trusted party, then message (`send-output`, sid) is ignored).

Figure 1: The functionality $\mathcal{F}_{adpt}^{OT}$ for OT with adaptively chosen inputs.

the conclusion of an execution of π, the honest party writes its output from that execution on its output tape.

In order to obtain token re-usability we will assume that once a party sends tokens to the other party, it also keeps some state information regarding the tokens. Therefore multiple executions of a protocol that recycle the same tokens will require some state information shared among the executions. Without such a requirement token re-usability would not be possible.

Security Definition. The security is defined as the emulation of the real execution in the ideal model. A protocol is secure if for every real-model adversary $\mathcal{A}$ and pair of input-selecting machines (M_1, M_2), there exists an ideal model adversary Sim such that for all initial inputs $\mathsf{init}_{\mathsf{S}}, \mathsf{init}_{\mathsf{R}}$, the outcome of an ideal execution with Sim is computationally indistinguishable from the outcome of a real protocol execution with $\mathcal{A}$. Notice that Sim knows the strategy used by the honest parties to choose their inputs. However, Sim does not know the initial input of the honest party, nor the random tape used by its input-selecting machine (any "secrets" used by the honest parties are included in the initial input, not the input-selecting machine).

Definition 2 (Security under adaptive-input sequential composition (adapted from [Lin04]))**.** *Let $\mathcal{F}_{adpt}^{OT}$ and π be as above. Protocol π is said to securely compute $\mathcal{F}_{adpt}^{OT}$ under sequential self composition if for every real-model non-uniform PPT adversary $\mathcal{A}$ controlling P_i ($i \in \{1, 2\}$) and every pair of PPT input-selecting machines $\overline{M} = (M_1, M_2)$, there exists an ideal-model non-uniform PPT adversary Sim controlling P_i, such that:*

$$\{\mathbf{Ideal}_{\mathcal{F}_{adpt}^{OT}, \mathsf{Sim}, \overline{M}}(\mathsf{n}, \mathsf{init}_{\mathsf{S}}, \mathsf{init}_{\mathsf{R}}, z)\}_{\{\mathsf{n} \in N; \mathsf{init}_{\mathsf{S}}, \mathsf{init}_{\mathsf{R}}, z \in \{0,1\}^*\}} \stackrel{c}{\equiv} \{\mathbf{Real}_{\pi, \mathcal{A}, \overline{M}}(\mathsf{n}, \mathsf{init}_{\mathsf{S}}, \mathsf{init}_{\mathsf{R}}, z)\}_{\{\mathsf{n} \in N; \mathsf{init}_{\mathsf{S}}, \mathsf{init}_{\mathsf{R}}, z \in \{0,1\}^*\}}.$$

Input/Output Privacy. The security definition stated above says that no PPT distinguisher provided of the pair of inputs (input of honest player and input of malicious player) and the pair of outputs (output of honest player and output of malicious player) is able to distinguish if they are coming from the simulated or from the real experiment. Such security definition can be referred to as input/privacy definition since given both input and output of the honest party any distinguisher is not able to tell apart the simulation from the real world. We refer as input privacy the restricted security notions in which the distinguisher is given the input of all the parties and the output of *only* the malicious party. Output privacy instead is achieved when security holds when the distinguisher is given the output of *both* honest and malicious parties. In our construction we achieve input/output privacy with respect to a malicious receiver. Concerning the malicious sender we achieve only input privacy (unconditionally). Output privacy against malicious sender is preserved if tokens are used for one execution. When token are reused the simulation does not go through and thus a distinguisher given the output of the honest party could tell apart the simulated experiment from the real one.

3 Efficient Non-Interactive 1-Out-Of-2 String Oblivious Transfer

In this section we show a protocol for efficient non-interactive OT with stateful tamper proof tokens. The protocol will use two tokens and will be secure under sequential composition. More specifically, input privacy will be protected against malicious adversaries, while output privacy can be compromised by a malicious sender. We first start with an high-level description of our result, and then we give the formal protocol and proof.

High-level description of techniques and protocol. In the protocol proposed by Kolesnikov in [Kol10] there is a first round where the sender creates and sends a token to the receiver. Then, the receiver interacts with the token and uses the obtained outputs to play two more rounds with the sender. The protocol is therefore interactive, which could be a limitation in several applications (e.g., when the sender broadcasts messages through a satellite and the receiver can not transmit messages back to the sender), and when a protocol is used as subprotocol (e.g., for protocol composition issues and round complexity optimality).

Our main technique to overcome this limitation, therefore obtaining a non-interactive protocol, consists in asking the sender S to send two tokens to the receiver R. Tokens will be used by R to play locally (i.e., without exchanging messages with the sender) an execution of OT.

S sends one-time pad encryptions of s_0 and s_1 computed with two independent keys k_0, k_1 given in output by two forward-secure PRGs. One of the two tokens, that we denote T_s, internally includes both keys. Then R will play a local OT with T_s in order to obtain one of the key used by S to encrypt the secret strings and therefore get s_i. We implement the local OT execution between T_s that will play the role of a sender having the two keys mentioned above as input, and R that is interested in getting k_i without revealing i to T_s. Here one can think that there is a circularity since we wanted to achieve efficient OT, but now we are back to the problem of achieving efficient OT. This is only partially true. Indeed, by using this technique, we have at least solved the problem with the round complexity. We have only one message sent by S in the communication network, and now we need an efficient OT that is played locally and thus does not need to be non-interactive. The local OT between T_s and R is implemented using again the help of hardware tokens. This is why we require in our solution a second token that we denote as T_k. The role of T_k is to help for the design of a 2-message efficient OT protocol. The local execution of OT goes as follows: T_s playing as a sender will keep the two keys (k_0, k_1) encrypted by means of two additional keys $\widehat{k_0}, \widehat{k_1}$. R will send a random bit b to T_s that will answer with $(\widehat{k_0} \oplus k_0, \widehat{k_1} \oplus k_1)$ if $b = 0$ and with $(\widehat{k_0} \oplus k_1, \widehat{k_1} \oplus k_0)$ if $b = 1$. Then, R sends $b \oplus i$

to T_k that will answer sending $\widehat{k_{b\oplus i}}$. It is easy to see that both T_s and T_k only see random bits, but however R obtains the correct key that allows it to obtain k_i first, and then s_i.

One could argue that it would have been sufficient to have only one token such that R provides his own input and obtains the key. This simple solution does not work. Indeed this solution allows the token to have a view containing all the inputs played by the receiver in all the previous executions. Therefore a malicious sender by means of this information can program a token such that when the input of the receiver corresponds to a specific pattern then the token must abort (or deviate from the honest behavior). When a pattern that is considered particular disadvantageous for the sender is detected by the token, the former can be instructed such that it must deviate from the protocol. In particular the deviation could consist of the abort of the token and thus the denial of the service. Thinking of a real application a provider of a service could be willing to cheat and thus to compromise his reputation with a customer when he has the guarantee that a disadvantageous scenario has been avoided. The use of two tokens breaks the guarantee of the pattern and introduces risks for the sender, meaning that the sender has to bet on the input of the receiver. Still, using two tokens, the probability that the sender can lead the output obtained by the receiver depending the input of R is not negligible, but now the sender is compromising his reputation without the guarantees that the customer fitted a particular pattern. Translated in a real scenario this means that a provider would not be willing to lose a costumer (and the reputation) if he is not sure that the costumer was approaching a very disadvantageous pattern for him. This in spirit has some similarities with the covert model of [AL07], even though technically there are some differences. We show a protocol enjoying this extra feature, even though we will not formally claim in this work an extra property as it would require a precise formalization of this security model, and we defer it to future research.

We now give the intuition behind the security proof. When playing as malicious sender the view of the adversarial sender is empty since the protocol is non-interactive. Moreover, the view of each malicious token sent by the adversarial sender only consists of random bits (i.e., the queries of the token). Thus input privacy is preserved unconditionally against a malicious sender. However, tokens can coordinate an attack that leads the receiver to abort depending on his previous inputs. This attack can not be replicated with the same probability by the simulator for the following reason. When simulating the honest receiver to a malicious sender (and thus malicious tokens) the simulator plays with random inputs and leaves the tokens in a random state such that also the joint view of the tokens is uniformly distributed. This deviates from the distribution of the joint view of the tokens left by the honest receiver that instead corresponds to the honest receiver's input. However, this is a threat only if the token is used more than once, indeed w.r.t one execution Sim challenges the tokens with both inputs in order to get both keys, thus any malicious behavior of the token is detected by the Sim with the same probability as the honest receiver.

When playing as malicious receiver, the adversarial view consists of the tuple containing encryptions of the two strings (obtained by the sender), encryptions of the two keys (obtained by T_s) and one decryption key (obtained by T_k) that will determine the final decrypted string.

Due to the use of the one-time-pad encryption, any tuple is such that it is always possible to have four values of the view that are completely random and to set the remaining value so that one would compute the output string s_i. This property, along with the ability of observing all queries made by an adversarial receiver to the tokens (this coordination is not possible between malicious tokens and the malicious sender because they can not communicate with each other), allows the simulator to answer with randomly chosen strings and equivocate the encryptions by setting the remaining value of the view properly.

3.1 Non-Interactive String OT

The formal protocol is depicted in Fig. 2.

Non-Interactive Initialization Phase:

S prepares and sends tokens to R as follows:

1. compute $K = \{(\mathsf{k}_0, \mathsf{St}_0) \leftarrow \mathsf{gen}_0.\mathsf{next}(\mathsf{GEN.key}(1^\mathsf{n})); \ (\mathsf{k}_1, \mathsf{St}_1) \leftarrow \mathsf{gen}_1.\mathsf{next}(\mathsf{GEN.key}(1^\mathsf{n}))\}$;
2. compute $\widehat{K} = \{(\widehat{\mathsf{k}_0}, \widehat{\mathsf{St}_0}) \leftarrow \widehat{\mathsf{gen}_0}.\mathsf{next}(\mathsf{GEN.key}(1^\mathsf{n})); \ (\widehat{\mathsf{k}_1}, \widehat{\mathsf{St}_1}) \leftarrow \widehat{\mathsf{gen}_1}.\mathsf{next}(\mathsf{GEN.key}(1^\mathsf{n}))\}$;
3. create token T_s seeded with the pair $(\widehat{K}, K)$ and token T_k seeded with $\widehat{K}$ such that:

   T_k on input a bit c:
   1. output $\widehat{\mathsf{k}_c}$;
   2. update keys: $(\widehat{\mathsf{k}_0}, \widehat{\mathsf{St}_0}) \leftarrow \widehat{\mathsf{gen}_0}.\mathsf{next}(\widehat{\mathsf{St}_0})$; $(\widehat{\mathsf{k}_1}, \widehat{\mathsf{St}_1}) \leftarrow \widehat{\mathsf{gen}_1}.\mathsf{next}(\widehat{\mathsf{St}_1})$.

   T_s on input a bit b:
   1. output $\widehat{e}_0 \leftarrow \widehat{\mathsf{k}_0} \oplus \mathsf{k}_b$; $\widehat{e}_1 \leftarrow \widehat{\mathsf{k}_1} \oplus \mathsf{k}_{(1-b)}$;
   2. update keys:
      $(\widehat{\mathsf{k}_0}, \widehat{\mathsf{St}_0}) \leftarrow \widehat{\mathsf{gen}_0}.\mathsf{next}(\widehat{\mathsf{St}_0})$; $(\widehat{\mathsf{k}_1}, \widehat{\mathsf{St}_1}) \leftarrow \widehat{\mathsf{gen}_1}.\mathsf{next}(\widehat{\mathsf{St}_1})$;
      $(\mathsf{k}_0, \mathsf{St}_0) \leftarrow \mathsf{gen}_0.\mathsf{next}(\mathsf{St}_0)$; $(\mathsf{k}_1, \mathsf{St}_1) \leftarrow \mathsf{gen}_1.\mathsf{next}(\mathsf{St}_1)$.
4. send tokens T_s and T_k to R.

Non-Interactive Oblivious Transfer.

S on input (s_0, s_1) works as follows:

1. compute e_0, e_1 as: $e_0 \leftarrow \mathsf{k}_0 \oplus s_0$; $e_1 \leftarrow \mathsf{k}_1 \oplus s_1$;
2. update keys: $(\mathsf{k}_0, \mathsf{St}_0) \leftarrow \mathsf{gen}_0.\mathsf{next}(\mathsf{St}_0)$; $(\mathsf{k}_1, \mathsf{St}_1) \leftarrow \mathsf{gen}_1.\mathsf{next}(\mathsf{St}_1)$;
3. send $\{e_0, e_1\}$ to R.

R on input i, upon receiving $\{e_0, e_1\}$ works as follows:

1. choose a random bit b and computes $c \leftarrow b \oplus i$;
2. query T_k with input c and obtains $\widehat{\mathsf{k}'_c}$, in case of abort output 0^n;
3. query T_s with input b and obtains $\widehat{e}'_0, \widehat{e}'_1$, in case of abort output 0^n;
4. compute $\mathsf{k}'_i \leftarrow \widehat{\mathsf{k}'_c} \oplus \widehat{e}'_c$;
5. compute $s'_i \leftarrow \mathsf{k}'_i \oplus e_i$ and return s'_i.

Figure 2: Efficient Non-Interactive String OT with 2 Tokens.

Theorem 1. *If* $\mathsf{GEN} = (\mathsf{GEN.key}, \mathsf{GEN.next}, \mathsf{n}, \mathsf{t})$ *is a forward-secure PRG then protocol 3.1 securely and efficiently implements* $\mathcal{F}_{adpt}^{OT}$ *functionality if tokens and not re-used, while when tokens are re-used output correctness is not guaranteed. Furthermore the protocol is forward secure.*

Proof. Correctness is straightforward and relies on the properties of the $\oplus$ operator. R with input i aims at decrypting s_i from e_i. Thus it needs to get the key k_i by interrogating tokens T_k and T_s. Depending on the random bit b chosen for the query to T_s, due to token's correctness, the desired key k_i is hidden in $\widehat{e}'_{i\oplus b}$ through an OTP encryption with $\widehat{\mathsf{k}_{i\oplus b}}$. Hence, by querying

the token T_k with input $i \oplus b$, R retrieves the correct key $\widehat{\mathsf{k}_{i\oplus b}}$. Finally, by the properties of the $\oplus$ operator, from $\widehat{e}_{i\oplus b}$ using $\widehat{\mathsf{k}_{i\oplus b}}$, R gets k'_i and it uses k'_i to obtain s_i from e_i.

We now show the ideal world adversary Sim, by considering the following cases.

Case 1: sender is corrupted. Let $\mathcal{A}$, $\mathcal{A}_{\mathsf{T}_s}$ and $\mathcal{A}_{\mathsf{T}_k}$ be PPT adversaries controlling S, T_s and T_k respectively. We show a PPT ideal world adversary Sim that simulates the honest receiver R in the real world experiment to the adversary $\mathcal{A}$ in order to extracts the pair (s_0, s_1) and be able to carry out the same attack in the ideal world. Sim runs $\mathcal{A}$ internally and works as follows. It obtains tokens $\mathcal{A}_{\mathsf{T}_s}$ and $\mathcal{A}_{\mathsf{T}_k}$. Upon receiving the *start-session* message from $\mathcal{A}$, Sim sends the message ($\texttt{start session}$) to $\mathcal{F}_{adpt}^{OT}$. This leads the ideal world receiver to get the message ($\texttt{start session}$, sid) from $\mathcal{F}_{adpt}^{OT}$ and to activate the input-selecting machine M_2 in order to obtain its next input. Upon receiving the pair (e_0, e_1) from $\mathcal{A}$, Sim interrogates token $\mathcal{A}_{\mathsf{T}_s}$ with a randomly chosen bit b and obtains the pair $(\widehat{k}_0 \oplus \mathsf{k}_b, \widehat{k}_1 \oplus \mathsf{k}_{1-b})$. Let st be the state of the simulation at this point. Sim then rewinds $\mathcal{A}_{\mathsf{T}_s}$ and queries it with bit $1-b$ obtaining $(\widehat{k}_0 \oplus \mathsf{k}_{1-b}, \widehat{k}_1 \oplus \mathsf{k}_b)$. The the simulator continues the execution from st (i.e., the rewind is played as a look-ahead and no query computed during the rewind will appear in the future view of $\mathcal{A}$). Note that it is crucial that the simulator chooses a random bit for the token's queries. Indeed if the simulator just queries the token with bit 0 and then bit 1 we have that the main thread of the simulation always includes 0 as query for T_s, which is clearly different from what the honest receiver sends in the real world. Finally Sim continues from state st (which is the main thread of the simulation), queries token $\mathcal{A}_{\mathsf{T}_k}$ with a randomly chosen bit c obtaining $\widehat{\mathsf{k}}_c$. At this point Sim gets both decryption keys and it can decrypt strings s_0, s_1 to play in the ideal world. Hence, Sim sends the message $(\texttt{sender}, \mathsf{sid}, (s_0, s_1))$ followed by $(\texttt{send-output}, \mathsf{sid})$ message to $\mathcal{F}_{adpt}^{OT}$. Note that the simulator can fail the decryption of both strings due to aborts in the following cases. In case T_k does not abort and T_s aborts then the simulator will not be able to extract at least one string to play in the ideal functionality. Thus it will play one of the following three pairs $(s_0, 0^\mathsf{n}), (0^\mathsf{n}, s_1)$ or $(0^\mathsf{n}, 0^\mathsf{n})$, depending on the query answered by T_s. In case T_k aborts then the simulator will play $(0^\mathsf{n}, 0^\mathsf{n})$ in the ideal world.

The intuition behind the proof of the input privacy of the receiver lies on the fact that all queries in the view of a token correspond to uniformly chosen random bits while the view of S, due to the non-interactiveness, is void. The output privacy is compromised when the token is reused, the reason is explained in details in section 3.1.

Case 2: receiver is corrupted. In this case we show a PPT ideal world adversary Sim that simulates S in the real world in order to extract the bit from the malicious receiver $\mathcal{A}$ and carry out the same attack in the ideal world. Sim runs $\mathcal{A}$ internally and works as follows. Upon receiving the *start-session* message from $\mathcal{A}$, Sim sends the message ($\texttt{start session}$) to $\mathcal{F}_{adpt}^{OT}$. Then the ideal world sender will receive the message ($\texttt{start session}$, sid) from $\mathcal{F}_{adpt}^{OT}$ and it will activate its input-selecting machine M_1 to get the input to send to the functionality. At this point Sim randomly chooses the tuple $(e_0, e_1, \mathsf{t}_0, \mathsf{t}_1, \mathsf{r})$ and sends it to $\mathcal{A}$. Then Sim must answer to the queries made by the adversary to tokens T_s and T_k. We distinguish two cases. Case 2.1: the adversary queries T_s first. In this case, as the adversary interrogates T_s with input b, Sim outputs a randomly chosen pair of strings $\widehat{e}_0, \widehat{e}_1$. Then, when $\mathcal{A}$ queries the token T_k with input c, Sim computes $i \leftarrow c \oplus b$ and plays $(\texttt{receiver}, \mathsf{sid}, i)$ in the ideal functionality in order to get the string s_i. Now the simulator computes $\widehat{\mathsf{k}}_c \leftarrow \widehat{e}_c \oplus e_i \oplus s_i$ and returns it to $\mathcal{A}$. Case 2.2: the adversary queries T_k first. In this case, as the adversary interrogates T_k with input c, Sim outputs a randomly chosen string $\widehat{\mathsf{k}}_c$. Then, when $\mathcal{A}$ queries the token T_s with input b, Sim computes $i \leftarrow c \oplus b$ and plays $(\texttt{receiver}, \mathsf{sid}, i)$ in the ideal functionality in order to get the string s_i. Then the simulator sets $\widehat{e}_c \leftarrow e_i \oplus s_i \oplus \widehat{\mathsf{k}}_c$ and $\widehat{e}_{(1-c)}$ as a randomly chosen string,

and outputs $(\widehat{e}_0, \widehat{e}_1)$. Finally the simulator will send the message $(\texttt{send-output}, \mathsf{sid})$ in the ideal world. Note that, the case where $\mathcal{A}$ queries tokens many times for the same execution (i.e., without having received new pairs from the sender) is not harmful since Sim is only required to choose random keys to answer the queries consistently for both tokens and records the keys according to the order they were queried to make consistent the future messages to be sent. The case in which $\mathcal{A}$ first queries one token, then gets the message from Sim and then queries the last token is managed following the previous approach (only the order with which Sim has to react to $\mathcal{A}$'s behavior changes). The last case regards Sim that gets the string from the ideal execution before actually sending the pair (e_0, e_1) to $\mathcal{A}$. However, this can be easily managed as Sim can again compute (e_0, e_1) properly, so that $\mathcal{A}$ gets $s_{b\oplus c}$. Finally, when $\mathcal{A}$ performs token corruption, Sim chooses random values $K = (\mathsf{k}_0, \mathsf{k}_1, \mathsf{St}_0, \mathsf{St}_1), \widehat{K} = (\widehat{\mathsf{k}}_0, \widehat{\mathsf{St}}_0, \widehat{\mathsf{k}}_1, \widehat{\mathsf{St}}_1)$ and returns the value according to the token that was corrupted. We finally stress that after corruption of a token, no other OT has to be simulated w.r.t. the sender that sent that token.

The simulator runs in polynomial time. When playing as receiver Sim is required to perform the rewind of a token. Since the malicious sender's attack (that involves $\mathcal{A}, \mathcal{A}_{\mathsf{T}_s}, \mathcal{A}_{\mathsf{T}_k}$) runs in polynomial time, then interacting with a token requires polynomial time and thus the simulator interacting with the token twice will be still polynomial. Notice that a rewind does not involve to repeat work done for other sessions, and thus there is no blow up in the running time of the simulator. Now consider the case of a malicious sender that starts sequentially many executions with different receivers. Since the protocol is non-interactive (a malicious sender has no message in his view) and the extraction process of the simulator involves only rewinding of tokens (and not of the sender) each execution can be resolved independently of the others, and thus this part can be simulated in polynomial time. When playing as a sender the simulator is required to extract the bit of the receiver. The trick is that in this case the simulator has complete control on the tokens, and can see both queries before answering to the second query (this can not be done by a malicious sender). Using the properties of the $\oplus$ operator, it can disclose at its wish the previously sent message by answering properly to the queries. Thus each execution requires only straight-line simulation, which is obviously done in polynomial time.

Achieving adaptive-input property. The adaptive-input property allows players to choose the next input to be provided in the protocol/functionality adaptively on the outputs obtained previously. Proving security in this setting requires a simulator able to reproduce the adversarial view without leading players of the simulated experiment to initiate more protocol executions (and thus more queries to the input-selecting machine) then the real world experiment. This could happen when the simulator's extraction strategy relies on the rewind of the malicious player. In such a case a malicious player could coordinate the executions and choose its inputs so that a simulator needs to open more ideal-word execution in order gets the inputs needed to simulate the adversarial view. This would make a deviation between real world and ideal world.

However, our simulator is not affected by such a problem, since it never rewinds the sender and the receiver, and the simulation is based on rewinding tokens in one case and in coordinating the answers to queries in another cases. Rewinding a token does not allow the adversary to open new sessions, and thus the input selecting machines do not need extra executions in the ideal world.

Indistinguishability of the views. We informally sketch the proof of security showing that the view of the adversary interacting with Sim is indistinguishable form the view of $\mathcal{A}$ interacting with the honest players. In our setting the adversary can decide to play as a malicious sender in all sessions or as a malicious receiver in all sessions. We will therefore show that against a

malicious receiver the simulation goes through while in case of the malicious sender it does only with respect to stand alone execution.

$\mathcal{A}$ is a malicious sender. *Stand-alone execution*: In this case the view of the malicious sender is void while the view of the malicious token is made by the bits of the queries. The honest receiver queries the token with randomly chosen bits. The simulator deviates from the honest receiver only for the rewind of the token. However after the rewind the token sees again a random bit. Therefore, the view of T_s and T_r interacting with Sim is distributed identically to when they are interacting with the honest receiver. This implies that for a stand alone execution our protocol achieves unconditional security against malicious senders.

Sequential composition: Reusing the same tokens across several executions arises subtle issues. First, observe that even if the view of the single token is uniformly distributed, the *joint* view of the tokens contains exactly the receiver's secret input. Observe also that, in order to get the decryption key the receiver has to combine the output of the tokens. Putting these two things together, we can have malicious tokens that coordinate their output such that the combined output is dependent on the input played by the receiver in a previous execution. As an example consider the following attack in which tokens lead R to abort if in the first execution R has played the bit 0. The sender sends malicious tokens that play honestly the first protocol execution. Then, in the second execution, each token aborts if the bit seen in the first execution is 1. Let us assume wlog that R played the first session with input 0. Then only two cases are possible: either R queried the tokens with the input $(0, 0)$ or with the input $(1, 1)$. Therefore in the second execution R aborts with probability $1/2$. In the simulated experiment, Sim playing with random inputs will play the first execution with input 0 with probability $1/2$, thus Sim has played with one the four possible pairs: $(0, 0), (0, 1), (1, 0), (1, 1)$. Thus, in the simulated experiment Sim aborts with probability $3/4$.

Note that this is a threat only if the token is used more than once, indeed w.r.t one execution Sim challenges the tokens with both inputs in order to get both keys, thus any malicious behavior of the token is detected by the Sim with the same probability of the honest receiver. The problem comes in the simulation of the subsequent executions. Indeed, after rewinding the token, the simulator might leave the token in a state that differs from the state left by the honest receiver. Therefore, the input of the receiver can be distributed according to a distribution that differs from the uniform distribution (that instead is used by the simulator when leaving bits in the views of the tokens). Summing up, with respect to token reusability, our protocol enjoys unconditional input privacy, while output privacy is not guaranteed.

$\mathcal{A}$ is a malicious receiver. In this case the view of the malicious receiver in one protocol execution consists of the pair of messages sent by the sender and the outputs of the tokens. Furthermore, dealing with a malicious receiver we have to prove that views remain indistinguishable even after the adversary gets the token's state performing the corruption. We notice that the simulator deviates from the honest sender in two ways: 1) it does not use GEN to perform the encryptions and the answers of tokens; 2) it sends the encryptions and it answers the token's queries with randomly chosen values except for one (depending on the order of the queries made to the tokens by the $\mathcal{A}$) that is properly set allowing the adversary to decrypt its selected string. For lack of space, a detailed sequence of hybrid arguments, showing that the view of the adversary in the real world does not change during the ideal world, is shown in Appendix A. We now informally argue the two main steps that characterize the hybrid arguments. In the first step, starting from the real world execution we replace the use of GEN to obtain the keys, by selecting randomly chosen keys. Moreover, once the adversary performs the corruption of the token the simulator returns randomly chosen values. It follows from the forward-security of GEN that the

view of $\mathcal{A}$ interacting with the sender can not be distinguished in polynomial time from the view of $\mathcal{A}$ interacting in this experiment. In the second step we show that the strategy used by Sim, answering queries and setting the last values properly is perfectly indistinguishable from the previous experiment. Indeed, this is due to the perfect security of OTP, since in both cases (i.e., the above experiment and during the simulation) all keys are used once and have uniform distribution. Summing up, we conclude that the view generated by the malicious receiver $\mathcal{A}$ playing with the honest sender S is computational indistinguishable from the view generated by $\mathcal{A}$ interacting with the simulator Sim. □

4 Acknowledgments

The work described in this paper has been supported in part by the European Commission through the ICT program under contract 216676 ECRYPT II and 215270 FRONTS, and in part by the MIUR Project PRIN "PEPPER: Privacy E Protezione di dati PERsonali" (prot. 2008SY2PH4).

We wish to thank the participants of the MAYA-ECRYPT II meeting that took place in IBM Research Zurich in May 2010, for several interesting discussions on cryptographic protocols with tamper proof hardware tokens. Moreover we thank Nico Döttling, Daniel Kraschewski and Jörn Müller-Quade for having pointed out the output correctness issue on a previous version of this work.

References

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: efficient protocols for realistic adversaries. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 137–156, Berlin, Heidelberg, 2007. Springer-Verlag.

[BGKW88] Michael BenOr, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 113–131, New York, NY, USA, 1988. ACM.

[BY03] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, April 13–17, 2003. Springer, Berlin, Germany.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 545–562, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[CLOS02] Ron Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, Lecture Notes in Computer Science, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[DKMQ11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, Berlin, Heidelberg, 2011. Springer-Verlag. To appear.

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

[FPS+11] Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider, and Ivan Visconti. Secure set intersection with untrusted hardware tokens. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 1–16, San Francisco, CA, USA, February 14–18 2011. Springer, Berlin, Germany.

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *Advances in Cryptology – CRYPTO 2010*, Lecture Notes in Computer Science, pages 173–190, Santa Barbara, CA, USA, August 2010. Springer, Berlin, Germany.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *Proceedings of the 28th Annual conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 39–56, Berlin, Heidelberg, 2008. Springer-Verlag.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 43:431–473, 1996.

[HL08] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standard smartcards. In *Computer and Communications Security (CCS'08)*, pages 491–500. Association for Computing Machinery, 2008. To appear in Journal of Cryptology.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Berlin, Germany.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007. Springer, Berlin, Germany.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th Annual ACM Symposium on Theory of Computing*, pages 20–31, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[Kol10] Vladimir Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 327–342, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes*

in Computer Science, pages 203–222, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany.

[MN05] Tal Moran and Moni Naor. Basing cryptographic protocols on tamper-evident seals. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005: 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 285–297, Lisbon, Portugal, July 11–15, 2005. Springer, Berlin, Germany.

[MS08] Tal Moran and Gil Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 527–544, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

A Details on the Indistinguishability of the Views

In this section we provide a detailed proof of the indistinguishability of the views of the adversarial receiver $\mathcal{A}$ when playing against an honest sender and when playing during the simulation (Section 3.1 includes a sketch of this proof).

Let $m = p(\mathsf{n})$, where p is a polynomial, consider an adversary performing a total of m protocol executions with possibly different senders: $\mathsf{S}_1, \mathsf{S}_2, \ldots, \mathsf{S}_\mathsf{p}$ in any order. Finally consider that $\mathcal{A}$ ends its attack by making token corruption. Since the security experiment with each different sender is anyway concluded after the corruption of the corresponding token then wlog we assume that $\mathcal{A}$ ends corrupting all tokens. The proof continues by showing the indistinguishability of the following hybrids games.

H_0: in this hybrid Sim with initial input init_S will internally simulates a real world execution between $\mathcal{A}$ and S and outputs whatever $\mathcal{A}$ outputs. This is identical to $\mathbf{Real}_{\pi,\mathcal{A},\overline{M}}(\mathsf{n}, \mathsf{init}_\mathsf{S}, \mathsf{init}_\mathsf{R}, z)$.

$H_{1,1}$: in this hybrid Sim works exactly as in H_0 except that, when playing as the sender S_1, it replaces the use of gen_0 by giving in output randomly chosen keys when required (i.e., for computing (e_0, e_1) and for answering queries to the token T_s). Obviously the same random value will be used to replace the j-th output of gen_0 both when computing the j-th message and when answering to the j-th query of T_s.

Suppose that there exists a PPT distinguisher D distinguishing the views generated in hybrid H_0 and $H_{1,1}$ then we can construct a PPT distinguisher B for the forward-security of gen_0. We let q denote the number of protocol executions between $\mathcal{A}$ and sender S_1 ($\mathcal{A}$ performs corruption at the end of the q-th execution obtaining the state that would have been used in the $q+1$ execution). The algorithm B runs the external forward-security experiment and has to guess whether it is in $\mathbf{Exp}^{\text{fprg-0}}_{\mathsf{GEN}}$ or $\mathbf{Exp}^{\text{fprg-1}}_{\mathsf{GEN}}$. At the i-th iteration of this experiment, B obtains Out_i. B internally runs the experiment played by honest senders and $\mathcal{A}$ except when simulating the i-th (for $i = 1, \ldots, q$) protocol execution between $\mathcal{A}$ and S_1. In this case it replaces the key generated by gen_0 with Out_i (keeping the usual consistency, it will use Out_i also for the i-th answer of T_s) and returns find to the external experiment obtaining the next output block Out_{i+1}. Then, after receiving the block Out_{q+1} in the last interaction, B returns guess to the external experiment and obtain the state St_{q+1}. Finally, when adversary $\mathcal{A}$ performs the corruption of token T_s, B

answers to $\mathcal{A}$ by sending the pair (St_{q+1}, Out_{q+1}) obtained from the external experiment, along with the state of gen_1. At the end of the simulation B provides the view of $\mathcal{A}$ to D and outputs whatever D outputs. Now if the sequence of Out_i was computed by the pseudo-random generator GEN, then the view generated by B is distributed identically to experiment H_0, otherwise if it was randomly chosen, the view is distributed identically to experiment $H_{1,1}$. By the forward security of GEN, H_0 and $H_{1,1}$ are computational indistinguishable.

$H_{1,2}, \ldots, H_{1,\mathsf{p}}$: in the hybrid $H_{1,j}$ (for $j = 2, \ldots, \mathsf{p}$), Sim works exactly as in the hybrid $H_{1,j-1}$ except that, when playing as the honest sender S_j, it replaces the use of gen_0 by selecting randomly chosen keys when required (keeping the usual consistency, it will use Out_i also for the i-th answer of T_s). By the same arguments as above it follows that hybrids $H_{1,j-1}$ and $H_{1,j}$ are computationally indistinguishable.

$H_{2,1}$: in this hybrid, Sim works exactly as in $H_{1,\mathsf{p}}$ except that, when playing as the sender S_1, it replaces the use of gen_1 by outputting randomly chosen keys when required (i.e., for computing (e_0, e_1) and for answering queries to the token T_s). Obviously the same random value will be used to replace the j-th output of gen_1 both when computing the j-th message and when answering to the j-th query of T_s. Precisely as we showed the indistinguishability of H_0 and $H_{1,1}$, it follows that $H_{1,\mathsf{p}}$ and $H_{2,1}$ are computationally indistinguishable.

$H_{2,2}, \ldots, H_{2,\mathsf{p}}$: in the hybrid $H_{2,j}$ (for $j = 2, \ldots, \mathsf{p}$), Sim works exactly as in the hybrid $H_{2,j-1}$ except that, when playing as the honest sender S_j, it replaces the use of gen_1 by selecting randomly chosen keys when required (keeping the usual consistency, it will use Out_i also for the i-th answer of T_s). Precisely as we showed the indistinguishability of $H_{1,j-1}$ and $H_{1,j}$, it follows that $H_{2,j-1}$ and $H_{2,j}$ are computationally indistinguishable.

$H_{3,1}$: in this hybrid, Sim works exactly as in $H_{2,\mathsf{p}}$ except that, when playing as the sender S_1, it replaces the use of $\widehat{\mathsf{gen}}_0$ by selecting randomly chosen keys when required (i.e., for answering queries made to tokens T_s and T_k). Precisely as we showed the indistinguishability of H_0 and $H_{1,1}$, it follows that $H_{2,\mathsf{p}}$ and $H_{3,1}$ are computationally indistinguishable.

$H_{3,2}, \ldots, H_{3,\mathsf{p}}$: in the hybrid $H_{3,j}$ (for $j = 2, \ldots, \mathsf{p}$), Sim works exactly as in the hybrid $H_{3,j-1}$ except that, when playing as the honest sender S_j, it replaces the use of $\widehat{\mathsf{gen}}_0$ by selecting randomly chosen keys when required (keeping the usual consistency, it will use Out_i for the i-th answers of T_s and T_k). Precisely as we showed the indistinguishability of $H_{1,j-1}$ and $H_{1,j}$, it follows that $H_{3,j-1}$ and $H_{3,j}$ are computationally indistinguishable.

$H_{4,1}$: in this hybrid, Sim works exactly as in $H_{3,\mathsf{p}}$ except that, when playing as the sender S_1, it replaces the use of $\widehat{\mathsf{gen}}_1$ by selecting randomly chosen keys when required (i.e., for answering queries made to T_s and T_k). Precisely as we showed the indistinguishability of H_0 and $H_{1,1}$, it follows that $H_{3,\mathsf{p}}$ and $H_{4,1}$ are computationally indistinguishable.

$H_{4,2} \ldots, H_{4,\mathsf{p}}$: in the hybrid $H_{4,j}$ (for $j = 2, \ldots, \mathsf{p}$), Sim works exactly as in the hybrid $H_{4,j-1}$ except that, when playing as the honest sender S_j, it replaces the use of $\widehat{\mathsf{gen}}_1$ by selecting randomly chosen keys when required. Precisely as we showed the indistinguishability of $H_{1,j-1}$ and $H_{1,j}$, it follows that $H_{4,j-1}$ and $H_{4,j}$ are computationally indistinguishable.

$H_{5,1}$: in this hybrid, Sim works exactly as in hybrid $H_{4,\mathsf{p}}$ except that in the first session Sim plays (e_0, e_1) and answers to the tokens queries sending $\widehat{k}_c$ and $(\widehat{e}_0, \widehat{e}_1)$ according to the description of the simulator (i.e., all values are random except the last one that is properly computed so that the adversary obtains $s_{b \oplus c}$ where b and c are the bits of the two queries). We stress that in $H_{4,\mathsf{p}}$ all keys used for such a session are random, and Sim sends to the adversary the following messages:

1. $e_0 = s_0 \oplus k_0$, where k_0 is a random key, as first part of the non-interactive message;

2. $e_1 = s_1 \oplus k_1$, where k_1 is a random key, as second part of the non-interactive message;
3. $\widehat{e}_0 = \widehat{k}_0 \oplus k_b$, where $\widehat{k}_0$ is a random key, as first part of the answer of T_s to query b;
4. $\widehat{e}_1 = \widehat{k}_1 \oplus k_{1-b}$, where $\widehat{k}_1$ is a random key, as second part of the answer of T_s to query b;
5. $\widehat{k}_c$ as answer of T_k to query c.

Notice that the adversary does not obtain neither $k_{1-(b\oplus c)}$ nor $\widehat{k_{(1-c)}}$. From the above equations this means that in the view of the adversary $e_{1-(b\oplus c)}$ and $\widehat{e}_{1-c}$ are random strings. The remaining values obtained by the adversary are $e_{b\oplus c}, \widehat{e}_c$ and k_c where the last two strings are random and the first string corresponds to $s_{b\oplus c} \oplus \widehat{e}_c \oplus k_c$.

In $H_{5,1}$, according to the description of the simulator, depending on $\mathcal{A}$'s behavior, we should distinguish the following cases.

Case 1: first Sim sends (e_0, e_1), then it plays as T_s and finally it plays as T_k.
Case 2: first Sim sends (e_0, e_1), then it plays as T_k and finally it plays as T_s.
Case 3: first Sim plays as T_k, then it sends (e_0, e_1) and finally it plays as T_s.
Case 4: first Sim plays as T_k, then it plays as T_s and finally it sends (e_0, e_1).
Case 5: first Sim plays as T_s, then it sends (e_0, e_1) and finally it plays as T_k.
Case 6: first Sim plays as T_s, then it plays as T_k and finally it sends (e_0, e_1).

We now show that the output of $H_{5,1}$ in **Case 1** is perfectly indistinguishable from the one of $H_{4,\mathsf{p}}$, the other cases follow the same approach and trivially fit the equations.

So we will assume that first Sim sends (e_0, e_1), then it plays as T_s and finally it plays as T_k. Notice that (e_0, e_1) will be played as two random strings, as well as $\widehat{e}_0$ and $\widehat{e}_1$ as answer to a query b of token T_s. Then the answer given by T_k to a query c will be $s_{b\oplus c} \oplus e_{b\oplus c} \oplus \widehat{e}_c$.

Now observe that again, as discussed previously for $H_{4,\mathsf{p}}$, the adversary does not obtain neither $k_{1-(b\oplus c)}$ nor $\widehat{k_{(1-c)}}$. Therefore in the view of the adversary both $e_{1-(b\oplus c)}$ and $\widehat{e}_{1-c}$ are random strings. Also here, it happens that the remaining values obtained by the adversary are $e_{b\oplus c}, \widehat{e}_c$ and k_c where the last two strings are random, indeed k_c has been computed as the $\oplus$ operation over three strings, of which two were random. Moreover the first string corresponds to $s_{b\oplus c} \oplus \widehat{e}_c \oplus k_c$ which is precisely what happens in $H_{4,\mathsf{p}}$.

Therefore the views are identically distributed.

$H_{5,2}, \ldots, H_{5,m}$: in the hybrid $H_{5,j}$, (for $j = 2, \ldots, m$), Sim works exactly as in the hybrid $H_{5,j-1}$ except that in the j-th session Sim computes the pair (e_0, e_1) and the answers of T_s and T_k as described above. By the same argument as before hybrids $H_{5,j}$ $H_{5,j+1}$ are identically distributed.

Since $H_{5,m}$ corresponds to the simulated game, the claim holds.