Secure Set Intersection with Untrusted Hardware Tokens

Marc Fischlin[1], Benny Pinkas[2], Ahmad-Reza Sadeghi[1,3], Thomas Schneider[3], and Ivan Visconti[4]

[1] Darmstadt University of Technology, Germany
marc.fischlin@gmail.com
[2] Bar Ilan University, Ramat Gan, Israel
benny@pinkas.net
[3] Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de
[4] University of Salerno, Italy
visconti@dia.unisa.it

Abstract. Secure set intersection protocols are the core building block for a manifold of privacy-preserving applications.

In a recent work, Hazay and Lindell (ACM CCS 2008) introduced the idea of using trusted hardware tokens for the set intersection problem, devising protocols which improve over previous (in the standard model of two-party computation) protocols in terms of efficiency and secure composition. Their protocol uses only a linear number of symmetric-key computations and the amount of data stored in the token does not depend on the sizes of the sets. The security proof of the protocol is in the universal composability model and is based on the strong assumption that the token is trusted by *both* parties.

In this paper we revisit the idea and model of hardware-based secure set intersection, and in particular consider a setting where tokens are not necessarily trusted by both participants to additionally cover threats like side channel attacks, firmware trapdoors and malicious hardware. Our protocols are very efficient and achieve the same level of security as those by Hazay and Lindell for trusted tokens. For untrusted tokens, our protocols ensure privacy against malicious adversaries, and correctness facing covert adversaries.

Keywords: cryptographic protocols, set intersection, untrusted hardware.

1 Introduction

A variety of applications with sophisticated privacy requirements can be based on secure set operations, in particular secure set intersection. Examples are versatile and range from government agencies comparing their databases of suspects on a national and international basis, to competing enterprises evaluating their performance on various aspects (items, deployed processes), to dating services.

The underlying protocols typically involve two mistrusting parties who compute an intersection of their respective sets (or some function of them). As we elaborate in §1.1 on related work, cryptographic research has proposed several solutions to this problem, each having its own strengths and weaknesses; in particular, the efficiency aspect is crucial for deployment in real-life scenarios: While software-based solutions use expensive public-key operations, it is also possible to incorporate a tamper-proof hardware token into the protocol, yielding more efficient schemes and/or avoiding impossibility results. However, this hardware-based model requires a strong trust model, i.e., a token trusted by all parties.

Background. In this paper we will focus on a recent proposal by Hazay and Lindell [1] that aims to design *truly practical* and secure set intersection protocols by introducing a new party, a (tamper-proof) hardware token $\mathcal{T}$. Here, one party, called the issuer $\mathcal{A}$, programs a key into the token $\mathcal{T}$ which protects this key from being accessible by the other party $\mathcal{B}$. At the same time, the manufacturer of the token ensures that the token correctly computes the intended function, i.e., $\mathcal{A}$ can only choose the secret key but cannot interfere with the token's program. The protocol is very efficient and requires the involved parties and the token to perform a few pseudorandom permutation evaluations, thus disposing of any public-key operations and/or random oracles as in previous efforts (cf. §1.1).

The use of the token in [1] is justified when trusted hardware manufacturers are available (e.g., manufacturers which produce high-end smartcards that have FIPS 140-2, level 3 or 4 certification). The security of the scheme is proven in the Universal Composability (UC) model [2], guaranteeing security even when composed with other protocols. It is important to note that today's high-end smartcards may have a sufficient amount of resources for executing the entire ideal functionality in a relatively simple use-case such as set intersection, although probably not on relatively large inputs. However, doing so would require to program the smartcard to implement this specific functionality. The protocols of [1] as well as the protocols we propose, on the other hand, can be run in practice by using cheap smartcards: they assume limited computation capabilities (only symmetric-key operations) and *constant* storage (see also [1]).

Motivation. The security proof of the scheme of [1] considers the universal composability framework inherently relying on the trustworthiness of the token, since it is assumed that both parties fully trust the token. This assumption, though, is critical with regard to several aspects regarding to what level tokens can be trusted in practice.

First, even extensive testing of the token cannot provide protection against errors and backdoors, introduced accidentally or deliberately in the underlying hardware and software stack running on it. A well-known example is the "Pentium bug" which caused the floating point division unit of the Intel Pentium$^{\text{TM}}$ processor to compute slightly wrong results for specific inputs [3]. Such flaws in the hardware can be exploited in so called "bug attacks" [4] to break the security of the underlying protocol. Moreover, although appropriate certification might help to ensure, to some degree, that at least the design of the token is backdoor-free, it is still unclear how to protect against hardware Trojans being maliciously introduced into the hardware during the manufacturing process, particularly because chip production is increasingly outsourced to other countries which are potentially untrusted or have their own evaluation standards.

Another threat concerns hardware and side-channel attacks allowing to break hardware protection mechanisms. Modern commercial smartcards have been equipped with a variety of measures to counter standard side-channel attacks. However, the severeness of attacks depends of course on the effort (see, e.g., the recently reported hardware attack on the Trusted Platform Module (TPM) [5]).

Our Contribution and Outline. After summarizing related works on set intersection and token-based protocols in §1.1, we introduce our setting and the employed primitives in §2, and review the basic protocol of [1] in §3. Afterwards, we present the following contributions.

We revisit the model of a fully trusted hardware token and provide several protocols for secure set intersection that make use of untrusted hardware tokens and fulfill different security targets. In our protocols only one party $\mathcal{A}$ trusts (some of) the hardware token(s) but the other party $\mathcal{B}$ does not. More concretely, we present a stepwise design of token-based set intersection protocols:

1. Guaranteeing the privacy of $\mathcal{B}$'s inputs in the *malicious* adversary model, using a *single* token trusted only by the issuer $\mathcal{A}$ (§4).
2. Additionally guaranteeing the correctness of $\mathcal{B}$'s outputs in the *covert* adversary model, using a *single* token trusted only by the issuer (§5).
3. Additionally preserving the privacy of $\mathcal{A}$'s inputs in the *malicious* adversary model, using *multiple* tokens of which at least one is trusted by issuer $\mathcal{A}$ (§6).

Moreover, our protocols have the "fall-back" security guarantees to the protocol of [1]: in case both parties fully trust the token, our protocols still provide the same security properties as [1]. While the original protocol of [1] does not provide any security guarantees in the case of untrusted token, our protocols achieve input privacy for malicious adversaries and output correctness for a covert token, i.e., any cheating attempt of the token may breach correctness (but not privacy) and is detectable with high probability.

1.1 Related Work

Set Intersection without Hardware Tokens. Several protocols for two-party set intersection secure in the *semi-honest* model have been proposed [6, 7, 8, 9, 10]. Protocols with security against *malicious* adversaries are given in [6,7,11,12,8,13,14,15,16,17]. A detailed summary and performance comparison of most of these protocols is given in [9]. Protocols with *covert* security are given in [12, 16]. All these protocols that do not employ hardware tokens need a non-negligible number of computationally expensive public-key operations [6]. In contrast, the protocols of [1] and our protocols perform a linear number of fast symmetric-key operations only.

Set Intersection with Hardware Tokens Trusted by Both Parties. HW tokens with limited capabilities that are trusted by both parties have been used to construct more efficient protocols for verifiable encryption and fair exchange [18], and secure function evaluation [19,20]. Additionally, government-issued signature cards have been proposed as setup assumption for UC [21]. Further, semi-honest tamper-proof hardware tokens can serve as basis for non-interactive oblivious transfer and hence non-interactive secure two-party computation, called one-time programs [22,23]. Our tokens need not to be trusted by both parties. In the rest of the paper we will extend the token-based set intersection model and protocol proposed recently in [1] which we summarize in §3.

Set Intersection with Hardware Tokens Trusted by the Issuer Only. HW tokens trusted by their issuer only were used as setup assumption for constructing UC commitments [24,25,26,27], and information-theoretic one-time programs [28]. These protocols use HW tokens merely to overcome known impossibility results, but do not claim to yield efficient protocols for practical applications.

To improve the performance of practical two-party secure function evaluation protocols, garbled circuits can be generated efficiently using a HW token trusted by its issuer only [29]. Furthermore, truly efficient oblivious transfer protocols with security against covert adversaries were proposed in [30]. We adapt techniques of [30] for constructing our protocols for secure set intersection.

2 Preliminaries

We denote the security parameter for symmetric schemes by t. A *pseudorandom permutation* (PRP) F is an algorithm which takes as input a key $k \in \{0,1\}^t$ and describes a "random-looking" permutation $F_k(\cdot)$ over $D = \{0,1\}^t$. If we drop the requirement on F being a permutation, then we have a *pseudorandom function* (PRF) instead. If it also holds that it is hard to distinguish permutation F_k from a random permutation given access to both the permutation and its inverse, then F is called a *strong* pseudorandom permutation (SPRP). Note that AES, for example, is believed to be a strong PRP.

2.1 The Setting for Token-Based Set Intersection Protocols

The general setting for the set intersection protocols we consider is as follows: Two parties, $\mathcal{A}$ and $\mathcal{B}$ would like to compute the intersection $\mathcal{F}_\cap(X,Y) = X \cap Y$ on their input sets $X = \{x_1, \ldots, x_{n_A}\}$ and $Y = \{y_1, \ldots, y_{n_B}\}$ such that only $\mathcal{B}$ obtains the output (while $\mathcal{A}$ learns nothing). Note that we assume that the set sizes are known to both parties. We further assume that elements from X and Y are from a domain $D = \{0,1\}^t$, i.e., $X, Y \subseteq D$. If needed, larger input data can be hashed to shorter strings with a collision-resistant hash function.

Our protocols have the following general structure: party $\mathcal{A}$ issues, i.e., buys, one or more hardware tokens $\mathcal{T}_1, \ldots, \mathcal{T}_n$, where $\mathcal{T}_i$ is manufactured by the hardware manufacturer $\mathcal{M}_i$. It initializes the tokens $\mathcal{T}_i$, and sends them to $\mathcal{B}$. In the case of protocols with a single token we simply call the token $\mathcal{T}$ and its manufacturer $\mathcal{M}$. In our model, any of the participating parties may be dishonest (where a dishonest token $\mathcal{T}$ refers to a maliciously produced token), and all malicious parties are controlled by a single adversary. We say that *a party trusts* $\mathcal{T}$ iff the other party cannot collude with $\mathcal{M}$ to produce a dishonest or breakable token. We consider static corruptions only.

To model hardware-based access we assume that, once a token is in possession of $\mathcal{B}$, $\mathcal{A}$ cannot communicate with the token anymore. In particular, the adversary may construct a malicious token, but may not interact with the token anymore, once it is sent to $\mathcal{B}$. The adversary can only communicate with the token through messages sent to and received from $\mathcal{B}$. Analogously, two tokens cannot communicate directly.

2.2 Security Models

While we denote by $\mathcal{A}, \mathcal{B}$, and $\mathcal{T}$ respectively the first (left) player, the second (right) player and the token, we will denote by $\mathcal{A}_I$ and $\mathcal{B}_I$ the players of the ideal world where parties just send their inputs to a set intersection functionality that then sends the intersection of the received inputs to $\mathcal{B}_I$.

We use different security notions. First, we consider *unconditional privacy* of the input of a player, i.e., regardless of the actions of the other malicious player, the input of an honest player will remain private in the sense that anything that can be computed about it can also be computed in the ideal world.

When we can carry a real-world attack mounted by an adversary during a protocol run into an ideal world attack, we achieve *simulation-based security.* If simulation cannot be achieved, we will instead downgrade to the weaker *indistinguishability-based security* notion. This last notion means that a malicious player cannot guess which input the other player has used during a protocol run, even when the honest player uses one of two inputs determined by the adversary.

The traditional notion of security through realizing an ideal functionality requires the simulation of any real-world attack into an ideal-world attack, and that the outputs of honest players do not deviate in the two worlds. We then say that the protocol *securely computes* (or *evaluates) the functionality* $\mathcal{F}_{\cap}(X, Y)$, and often specify the adversary's capabilities further, e.g., that the token is trusted or that it cannot be compromised by $\mathcal{B}$. This classical notion implicitly includes a *correctness* requirement: the output of a honest player depends only on its input and the implicit input used by the adversary in the protocol run.

When our protocols cannot achieve the *correctness* and *simulation* requirements simultaneously, we will downgrade the standard security notion to *covert security* [31], which means that the adversarial behavior can be detected by the honest player with some non-negligible probability ϵ, called the deterrence factor.[1] In all applications where the reputation of a player is more important than

[1] In addition the protocol must be *detection accurate* in the sense that in real-world executions no honest party accuses another honest party of cheating. All our protocols obey this property, albeit we do not mention this explicitly.

the output correctness of another player (e.g., where established enterprises offering services to citizens), this notion of covert security suffices, since there is a deterrence factor that discourages malicious actions.

We note that our protocols provide stronger security guarantees than security against the strongest notion of covert adversaries defined in [31], as no information about honest players' inputs is leaked, independently of whether cheating was detected or not. That is, in our case the ideal-world adversary can issue a `cheat` command (in case he wants to cheat) and this is announced to the parties with probability ϵ – but unlike in [31] the ideal-world adversary here does not get to learn the honest parties' inputs in case no cheat is announced. Still, in such a case we provide no correctness guarantee whatsoever.

3 Both Parties Trust Token [1]

We now review the model and protocol of [1]. Our models and protocols presented later extend on these to cope with untrusted hardware.

Model of [1]. In the model of [1], the hardware token $\mathcal{T}$ is assumed to honestly compute the intended functionality. The authors of [1] argue that this assumption is justified if highly trusted hardware manufacturers are available, e.g., manufacturers which produce high-end smartcards that have FIPS 140-2, level 3 or 4 certification. The token $\mathcal{T}$ is as reliable as its manufacturer $\mathcal{M}$ and, as only $\mathcal{T}$ is involved in the protocol but not $\mathcal{M}$, this security assumption is weaker than using $\mathcal{M}$ as a trusted third party.[2]

Set Intersection Protocol of [1]. The set intersection protocol of [1], depicted in Fig. 1, works as follows: In the setup phase, $\mathcal{A}$ initializes the HW token $\mathcal{T}$ with a random key k, a random message OK, and an upper bound on the size of $\mathcal{B}$'s input set n_B; $\mathcal{A}$ sends $\mathcal{T}$ to $\mathcal{B}$. In the online phase, $\mathcal{B}$ can query the token to evaluate F_k (where F is a SPRP as defined in §2) on each of its inputs. If $\mathcal{T}$ has been queried n_B times, it invalidates k (e.g., by deleting it)[3] and outputs OK to $\mathcal{B}$ who forwards it to $\mathcal{A}$. If OK is correct, $\mathcal{A}$ sends the evaluation of F_k on each of his inputs to $\mathcal{B}$. Finally, $\mathcal{B}$ computes the intersection by comparing the values obtained from $\mathcal{T}$ with those from $\mathcal{A}$. (Note that at that point $\mathcal{B}$ cannot query $\mathcal{T}$ anymore, i.e., all queries to $\mathcal{T}$ were independent of $\mathcal{A}$'s inputs.)

Security. According to Theorem 3 of [1], the above protocol UC-securely realizes the set intersection functionality when $\mathcal{T}$ is honest.

[2] This model is somewhat related to the common reference string (CRS) model in which a party trusted by all players generates a string according to a given distribution. The string is later used in the protocol. While a CRS is a static information generated before protocol executions, the trusted token will offer a trusted functionality *during* the execution of a protocol.

[3] This ensures that $\mathcal{B}$ gains no advantage when querying $\mathcal{T}$ in an invalid way.

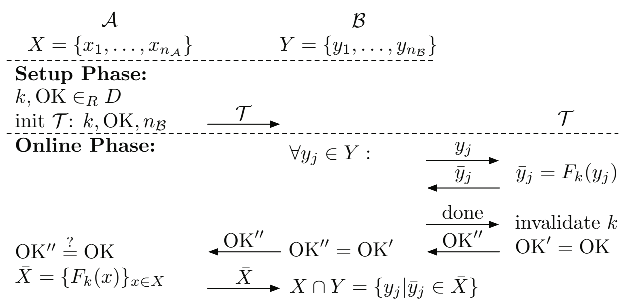

$\mathcal{A}$
$X = \{x_1, \ldots, x_{n_\mathcal{A}}\}$
$\mathcal{B}$
$Y = \{y_1, \ldots, y_{n_\mathcal{B}}\}$
Setup Phase:
$k, \mathrm{OK} \in_R D$
init $\mathcal{T}$: $k, \mathrm{OK}, n_\mathcal{B}$
$\mathcal{T}$
$\mathcal{T}$
Online Phase:
$\forall y_j \in Y$:
y_j
$\bar{y}_j$
$\bar{y}_j = F_k(y_j)$
done
invalidate k
$\mathrm{OK}'' \overset{?}{=} \mathrm{OK}$
OK''
$\mathrm{OK}'' = \mathrm{OK}'$
OK''
$\mathrm{OK}' = \mathrm{OK}$
$\bar{X} = \{F_k(x)\}_{x \in X}$
$\bar{X}$
$X \cap Y = \{y_j | \bar{y}_j \in \bar{X}\}$

Fig. 1. Set Intersection Protocol of [1]: token $\mathcal{T}$ is trusted by both parties

Efficiency. $\mathcal{T}$ performs $n_\mathcal{B}$ evaluations of F. The communication in the online phase contains the OK message from $\mathcal{B}$ to $\mathcal{A}$, and a message containing $n_\mathcal{A} t$ bits from $\mathcal{A}$ to $\mathcal{B}$. The overall online communication complexity is therefore $\mathcal{O}(n_\mathcal{A} t)$.

4 Only Issuer Trusts Token: Privacy of $\mathcal{B}$'s Input

The protocol of [1] assumes that $\mathcal{T}$ is fully trusted by both parties. Obviously, when one of the parties can break into $\mathcal{T}$ (e.g., by physical attacks or by colluding with its manufacturer $\mathcal{M}$), they can break the correctness or the privacy of the protocol. In the following we extend the protocol of [1] to make it non-interactive and guarantee privacy of $\mathcal{B}$'s inputs even if $\mathcal{A}$ and $\mathcal{T}$ are malicious.

Model. We consider the trust model where $\mathcal{B}$ does not trust $\mathcal{T}$ to behave correctly, i.e., $\mathcal{A}$ can collude with the hardware manufacturer $\mathcal{M}$ to produce a bad token $\mathcal{T}$. This model seems justified, as $\mathcal{B}$ is required to use a hardware token which is provided by $\mathcal{A}$, whom $\mathcal{B}$ might not trust.

Problem 1 ($\mathcal{A}$ colludes with $\mathcal{M}$ to break privacy of $\mathcal{B}$'s inputs). In the protocol of Fig. 1, the only message in which information about $\mathcal{B}$'s inputs can be leaked to $\mathcal{A}$ is the OK message. A corrupt player $\mathcal{A}$ can construct a corrupt token $\mathcal{T}$ that changes the OK message based on the inputs that $\mathcal{B}$ feeds to $\mathcal{T}$ (i.e., OK is used as covert channel), or $\mathcal{T}$ aborts the protocol (e.g., refuses to output OK).

Protocol. Problem 1 arises in the protocol of [1], as $\mathcal{B}$ first provides his input Y to $\mathcal{T}$, $\mathcal{T}$ answers $\mathcal{B}$ and finally $\mathcal{B}$ sends a message to $\mathcal{A}$ which depends on $\mathcal{T}$'s answer (OK). We eliminate this source of leakage from $\mathcal{T}$ to $\mathcal{A}$ in the protocol as shown in Fig. 2, by making the protocol non-interactive: First, $\mathcal{A}$ sends the permutations $\bar{X}$ of its inputs (as before). Afterwards, $\mathcal{B}$ obtains its permuted inputs $\bar{Y}$ from $\mathcal{T}$ by sending its inputs Y to $\mathcal{T}$. In contrast to the original protocol, $\mathcal{T}$ cannot reveal the permuted inputs $\bar{y}_j$ directly to $\mathcal{B}$ as otherwise $\mathcal{B}$, who already knows $\bar{X}$ now, could already compute parts of the intersection $X \cap \{y_1, \ldots, y_j\}$ and adaptively change his input depending on this. Instead, $\mathcal{T}$ encrypts each $\bar{y}_j$ by XORing it with a pseudo-random pad p_j which is derived by computing a pseudo-random function $f_s(j)$ keyed with a fixed secret key s. After having queried for all elements in Y, $\mathcal{B}$ has an encrypted copy of $\bar{Y}$. Now, $\mathcal{T}$ releases the pseudo-random pads p_j with which $\bar{Y}$ is encrypted to $\mathcal{B}$, who can finally recover $\bar{Y}$ and compute $X \cap Y$ as before.

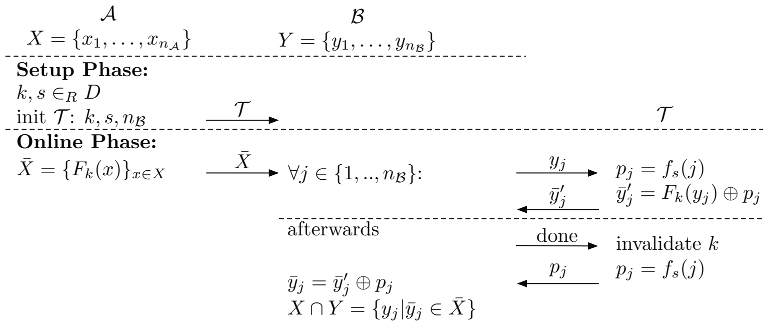

$\mathcal{A}$
$X = \{x_1, \ldots, x_{n_\mathcal{A}}\}$
$\mathcal{B}$
$Y = \{y_1, \ldots, y_{n_\mathcal{B}}\}$
Setup Phase:
$k, s \in_R D$
init $\mathcal{T}$: $k, s, n_\mathcal{B}$
$\mathcal{T}$
$\mathcal{T}$
Online Phase:
$\bar{X} = \{F_k(x)\}_{x \in X}$
$\bar{X}$
$\forall j \in \{1, .., n_\mathcal{B}\}$:
y_j
$\bar{y}'_j$
$p_j = f_s(j)$
$\bar{y}'_j = F_k(y_j) \oplus p_j$
afterwards
done
invalidate k
p_j
$p_j = f_s(j)$
$\bar{y}_j = \bar{y}'_j \oplus p_j$
$X \cap Y = \{y_j | \bar{y}_j \in \bar{X}\}$

Fig. 2. Set Intersection Protocol with Privacy of $\mathcal{B}$'s Inputs (Problem 1) w.r.t. malicious adversaries: token $\mathcal{T}$ is not trusted by $\mathcal{B}$

Theorem 1. *If F is a SPRP and f is a PRF, then the protocol depicted in Fig. 2:*

1. *securely evaluates $\mathcal{F}_\cap(X, Y)$ w.r.t. a malicious $\mathcal{B}$ that cannot break into $\mathcal{T}$;*
2. *keeps $\mathcal{B}$'s input unconditionally private in the indistinguishability sense w.r.t. a malicious $\mathcal{A}$;*
3. *securely evaluates $\mathcal{F}_\cap(X, Y)$ when both parties trust the token.*

Proof. To prove Theorem 1 we treat each corruption case separately.

$\mathcal{A}$ is corrupted and $\mathcal{T}$ is trusted by $\mathcal{A}$ and $\mathcal{B}$. As noted above, non-interactivity implies that $\mathcal{B}$'s input is protected unconditionally from a malicious $\mathcal{A}$. Here however, we can even prove unconditional security in a simulation-based sense, constructing an ideal-world adversary $\mathcal{A}_I$ that simulates in the ideal world the attack carried out by $\mathcal{A}$ in the real world. The difference here that allows us to achieve such a stronger security notion is that since the token is trusted, it has not been produced by $\mathcal{A}$, and therefore $\mathcal{A}$ has only black-box access to it. Thus, given a real-world adversary $\mathcal{A}$, we can construct an ideal-world adversary $\mathcal{A}_I$ that includes $\mathcal{A}$ and is able to read and write on its communication channels, including the ones that are supposed to be used for the communication with the token. Notice that since the token is trusted, from the fact that it answers to $\mathcal{B}$'s queries, it must be the case that $\mathcal{A}$ uploads to $\mathcal{T}$ both k and s – otherwise $\mathcal{T}$ would be in an inconsistent state and would not play with $\mathcal{B}$ (that therefore would just abort). Thus, $\mathcal{A}_I$ will obtain k and s from the initialization of the token performed by $\mathcal{A}$. Then, $\mathcal{A}_I$ reads the vector of messages $\bar{X}$ and inverts each $\bar{x}_j \in \bar{X}$ obtaining the original vector X that corresponds to the set that $\mathcal{A}$ would play in the real world. Then, $\mathcal{A}_I$ plays X in the ideal world. As a consequence,

the ideal-world honest player $\mathcal{B}_I$ will obtain the same input obtained by a real-world honest player $\mathcal{B}$, that plays the protocol with a trusted token. Finally $\mathcal{A}_I$ outputs whatever $\mathcal{A}$ outputs. As the joint distribution of the view of $\mathcal{A}$ and the output of $\mathcal{B}$ in real and ideal world are clearly identical, property 1 holds.

$\mathcal{A}$ is corrupted and $\mathcal{T}$ is trusted by $\mathcal{A}$ but not $\mathcal{B}$. Since the protocol is non-interactive, $\mathcal{A}$ does not get any message from $\mathcal{B}$ and therefore $\mathcal{B}$'s privacy is protected unconditionally. However, we cannot construct and ideal-world adversary $\mathcal{A}_I$ since we cannot extract $\mathcal{A}$'s input. Therefore we obtain unconditional indistinguishability of B's private input, and property 2 holds.

$\mathcal{B}$ is corrupted. To prove that $\mathcal{A}$'s input remains private in a simulation-based sense against a real-world malicious $\mathcal{B}$ we construct an ideal-world adversary $\mathcal{B}_I$ that internally simulates a protocol run to $\mathcal{B}$, extracts its input and plays the extracted input in the ideal world. $\mathcal{B}_I$ has control over the communication channels used by $\mathcal{B}$ to communicate with $\mathcal{T}$, and thus reads all queries y_j performed by $\mathcal{B}$, sending as answer random values $\bar{y}'_j$. Moreover, $\mathcal{B}_I$ sends to $\mathcal{B}$ a random vector $\bar{X}$ therefore simulating the message of the honest real-world $\mathcal{A}$. As soon as all elements of $\mathcal{B}$ have been sent to the (simulated) token, $\mathcal{B}_I$ groups all the elements in a set Y that is sent to the ideal functionality. $\mathcal{B}_I$ then obtains from the ideal functionality the intersection of Y with $\mathcal{A}_I$'s input, where $\mathcal{A}_I$ is the honest player of the ideal model. Let Z be the output of $\mathcal{B}_I$ in the ideal world. $\mathcal{B}_I$ now aims at giving Z to $\mathcal{B}$ in the internal execution of the real-world protocol. To do so, it performs the last n_B steps of the protocol sending values $p_1, \ldots, p_{n_B}$ as follows: if y_j is in Z then set $p_j = y'_j \oplus \bar{y}_j$, else set p_j equal to a random string. Then $\mathcal{B}_I$ outputs whatever $\mathcal{B}$ outputs.

Notice that the only difference in the view of $\mathcal{B}$ between the real-world and the simulated executions is that the former uses the SPRP F and the PRF f, while the latter uses random bits. We now show that any distinguisher between the two views, can be used to build either an adversary for F or an adversary f.

Consider the hybrid experiment G in which the real-world execution is played but F is replaced by random strings, still keeping consistency so that on the same input F produces the same output. Clearly G can be run in polynomial time and is computationally indistinguishable from the real-world execution, otherwise we have immediately a forgery for the SPRP F.

Consider now the next hybrid game G' in which all evaluations of f are replaced by random bits, still keeping consistency as above. Again, any distinguisher between G and G' would immediately produce a forgery for the PRF f.

Finally, consider the simulated execution of the real-world protocol. Both the message sent over the communication channel (i.e., $\bar{X}$) and the first bunch of answers of $\mathcal{T}$ (i.e., $\bar{y}'_j$) have the uniform distribution and are therefore identically distributed in both G' and in the simulated game. The final answers p_j received by $\mathcal{B}$ correspond in both the simulated game and in G' to random messages, with the only exception of the elements that appear in the intersection. In this last case the received messages p_j correspond precisely to the unique values that allow $\mathcal{B}$ to compute the values in the intersection. This holds both in G' and in the simulated execution. This allows us to conclude the proof of property 3. □

Efficiency and Token Reusability. While the round complexity of our protocol is optimal, compared to the 3 rounds of [1], its computational complexity is only by a factor of approximately 3 worse. Overall, the computational and storage requirements for $\mathcal{T}$ are the same in both protocols, namely symmetric-key operations (SPRP and PRF), and a small constant amount of secure storage.

Our protocols can be extended to reuse the same token for multiple protocol runs. For this, all information shared between $\mathcal{A}$ and $\mathcal{T}$ (i.e., the value k and s) is derived pseudo-randomly from a master-key known by $\mathcal{A}$ and $\mathcal{T}$ and some session identifier. The token $\mathcal{T}$ keeps track of the next session id using a strictly monotonic tamper-proof hardware counter which is available in most smartcards today. Also updating the usage counter $n_\mathcal{B}$ inside the token is possible via secure messaging as described in [1].

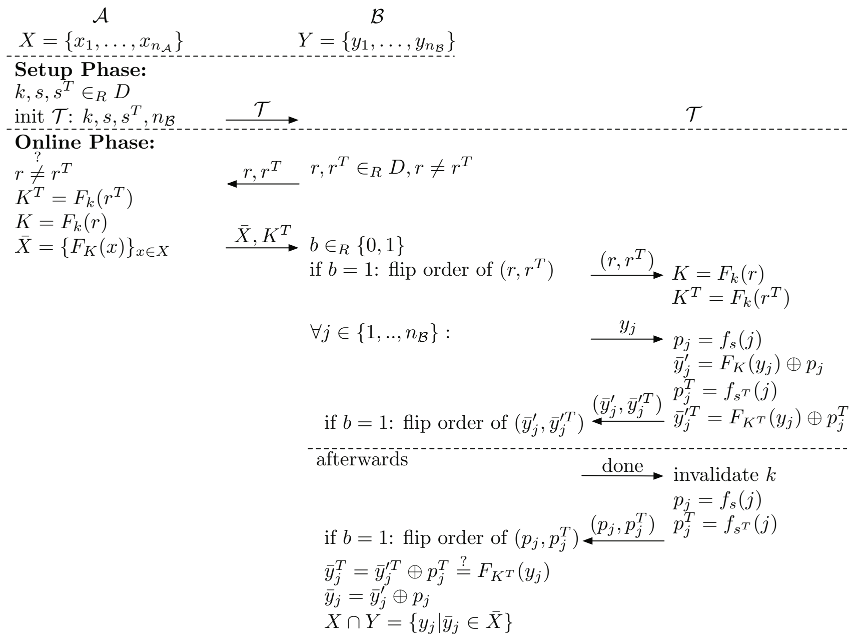
5 Only Issuer Trusts Token: Correctness of $\mathcal{B}$'s Output

In this section we extend the protocol of §4 to guarantee privacy and correctness when $\mathcal{B}$ does not trust the token. This is formalized by the following problem.

Problem 2 ($\mathcal{A}$ colludes with $\mathcal{M}$ to break correctness of $\mathcal{B}$'s output). In the protocols of Fig. 1 and Fig. 2, a corrupt $\mathcal{A}$ can enforce $\mathcal{B}$ to obtain in the protocol wrong outputs, i.e., different from $X \cap Y$: This can be done by creating a malicious token $\mathcal{T}$ that does not compute the permutation F correctly, but computes another function F' which maps multiple values to the same value or even depends on the history of values seen from $\mathcal{B}$.

Although Problem 2 does not affect the privacy of $\mathcal{B}$'s input, the correctness of $\mathcal{B}$'s output is no longer guaranteed. In many application scenarios this is not a problem, as a malicious $\mathcal{A}$ could also provide wrong inputs to the computation. However, a malicious token $\mathcal{T}$ could also compute a completely different function which does not correspond to set intersection at all: For example, a malicious $\mathcal{T}$ could output random values once it has obtained a value $y_i = 0$. In this case, the protocol computes some set $Z \subsetneq X \cap Y$ if $0 \in Y$, and $X \cap Y$ otherwise.

Protocol. We extend the protocol of Fig. 2 and adapt the oblivious transfer protocol of [30] to the set intersection scenario. We will therefore obtain both input privacy against malicious $\mathcal{A}$ and correctness against a covert $\mathcal{A}$ in the covert sense: $\mathcal{A}$ can actually succeed in violating the correctness of B's output with non-negligible probability but at the same time $\mathcal{B}$ can detect the cheating behavior of $\mathcal{A}$ with probability 1/2. The update of the protocol goes as follows: The basic idea is to let $\mathcal{T}$ compute two answers (using two different keys K, K^T), where $\mathcal{B}$ can verify the correctness of one answer ($\mathcal{B}$ obtains one key K^T from $\mathcal{A}$) without $\mathcal{T}$ knowing which one is verified. For this, $\mathcal{B}$ randomly chooses and sends to $\mathcal{A}$ a test value r^T and a distinct value r. Then, $\mathcal{B}$ obtains the test key $K^T = F_k(r^T)$ from $\mathcal{A}$, whereas the other key $K = F_k(r)$ remains unknown to $\mathcal{B}$ (to ensure this, $\mathcal{A}$ checks that $r^T \neq r$). Afterwards, $\mathcal{B}$ sends (r, r^T) to $\mathcal{T}$ in random order such that $\mathcal{T}$ can derive K, K^T without knowing which of them



$\mathcal{A}$
$\mathcal{B}$
$X = \{x_1, \ldots, x_{n_\mathcal{A}}\}$
$Y = \{y_1, \ldots, y_{n_\mathcal{B}}\}$
Setup Phase:
$k, s, s^T \in_R D$
init $\mathcal{T}$: $k, s, s^T, n_\mathcal{B}$
$\mathcal{T}$
$\mathcal{T}$
Online Phase:
$r \overset{?}{\neq} r^T$
r, r^T
$r, r^T \in_R D, r \neq r^T$
$K^T = F_k(r^T)$
$K = F_k(r)$
$\bar{X} = \{F_K(x)\}_{x \in X}$
$\bar{X}, K^T$
$b \in_R \{0,1\}$
if $b = 1$: flip order of (r, r^T)
(r, r^T)
$K = F_k(r)$
$K^T = F_k(r^T)$
$\forall j \in \{1, .., n_\mathcal{B}\}$:
y_j
$p_j = f_s(j)$
$\bar{y}'_j = F_K(y_j) \oplus p_j$
$p_j^T = f_{s^T}(j)$
$(\bar{y}'_j, \bar{y}'^T_j)$
if $b = 1$: flip order of $(\bar{y}'_j, \bar{y}'^T_j)$
$\bar{y}'^T_j = F_{K^T}(y_j) \oplus p_j^T$
afterwards
done
invalidate k
$p_j = f_s(j)$
$p_j^T = f_{s^T}(j)$
(p_j, p_j^T)
if $b = 1$: flip order of (p_j, p_j^T)
$\bar{y}_j^T = \bar{y}'^T_j \oplus p_j^T \overset{?}{=} F_{K^T}(y_j)$
$\bar{y}_j = \bar{y}'_j \oplus p_j$
$X \cap Y = \{y_j | \bar{y}_j \in \bar{X}\}$

Fig. 3. Set Intersection Protocol with Privacy of $\mathcal{B}$'s Input and (Covert) Correctness of $\mathcal{B}$'s Output when $\mathcal{T}$ is not trusted by $\mathcal{B}$, and Privacy of $\mathcal{A}$'s input when $\mathcal{A}$ trusts $\mathcal{T}$

is known to $\mathcal{B}$. Then, for each element $y_j \in Y$, $\mathcal{B}$ obtains $\bar{y}_j = F_K(y_j)$ and $\bar{y}_j^T = F_{K^T}(y_j)$ from $\mathcal{T}$ (after removing the pads p_j and p_j^T as in the protocol of Fig. 2). As $\mathcal{B}$ knows the test key K^T it can test the correctness of $\bar{y}_j^T$, whereas $\mathcal{T}$ can only guess whether to cheat on $\bar{y}_j$ or $\bar{y}_j^T$. Finally, $\mathcal{B}$ computes the intersection from $\bar{X}$ and $\bar{Y}$ as before.

The overall protocol shown in Fig. 3 provides $\mathcal{A}$ with input privacy against a malicious $\mathcal{B}$, which cannot break into the token, and provides $\mathcal{B}$ with input privacy (Problem 1) against a malicious $\mathcal{A}$ and $\mathcal{T}$ and output correctness against a covert $\mathcal{A}$ and $\mathcal{T}$ (Problem 2).

Theorem 2. *If F is a SPRP and f is a PRF, then the protocol depicted in Fig. 3:*

1. *securely evaluates $\mathcal{F}_\cap(X, Y)$ w.r.t. a malicious $\mathcal{B}$ that cannot break into $\mathcal{T}$;*
2. *securely evaluates $\mathcal{F}_\cap(X, Y)$ w.r.t. a covert $\mathcal{A}$ with deterrence factor $\epsilon = 1/2$;*
3. *securely evaluates $\mathcal{F}_\cap(X, Y)$ when both parties trust the token.*

$\mathcal{B}$'s input is still (unconditionally) private even w.r.t. malicious $\mathcal{A}$, as in Property 2 of Theorem 1.

Proof (Sketch). To prove Theorem 2 we consider each property individually.

Malicious $\mathcal{B}$ that cannot break into $\mathcal{T}$. We show an ideal world adversary $\mathcal{B}_I$. This adversary $\mathcal{B}_I$ internally runs $\mathcal{B}$ simulating also $\mathcal{T}$'s answers. $\mathcal{B}_I$ sends to $\mathcal{B}$

a random vector of messages $\bar{X}$ and a random key K^T. When simulating $\mathcal{T}$'s answers before `done`, B_I plays honestly when test queries are performed (i.e., using K_T for the test queries along with the pseudorandom function indexed by s_T) and sending random messages otherwise, as already done in the proof of Theorem 1. When message `done` has been received, B_I plays in the ideal world the input extracted from the queries received by $\mathcal{T}$ and gets back the intersection Z. Here B_I proceeds by computing values p_j^T honestly, but adaptively computing all final p_j values so that the view of $\mathcal{B}$ will still be computationally indistinguishable, precisely as in the proof of Theorem 1.

Note that, since $\mathcal{A}$ checks that $r \neq r^T$, the pseudorandom keys K and K^T are computationally independent, and can be essentially replaced by independent random keys. A straightforward hybrid game shows that by the pseudorandomness of F this does not change $\mathcal{B}$'s success probability significantly.

Covert $\mathcal{A}$*.* Informally, the privacy of $\mathcal{B}$'s input is preserved as $\mathcal{A}$ does not obtain any message from $\mathcal{B}$ besides the random values r, r^T. The same argument which has been applied already in the proof of Theorem 1 about protecting $\mathcal{B}$'s input from a malicious sender, applies here as well. The more interesting difference however consists now in proving correctness of $\mathcal{B}$'s output in the covert sense: showing that a success of $\mathcal{A}$ in violating the correctness of $\mathcal{B}$'s output can be detected by $\mathcal{B}$ with probability $\epsilon = 1/2$, and this is achieved through the cut-and-choose construction of [30].

To formally prove the correctness of $\mathcal{B}$'s output we build a simulator *Sim* which plays as an honest $\mathcal{B}$ against adversaries $Adv_{\mathcal{A}}$ and $Adv_{\mathcal{T}}$ who control $\mathcal{A}$ and $\mathcal{T}$, respectively. As the token is not necessarily honest and hence a cheating $Adv_{\mathcal{A}}$ does not need to initialize $\mathcal{T}$ at all, *Sim* cannot learn the token's keys k, s, s^T from the initialization message sent from $Adv_{\mathcal{A}}$ to $Adv_{\mathcal{T}}$. Instead, *Sim* determines whether the adversary cheats in the protocol as follows: *Sim* obtains both opening keys K^T and K from $Adv_{\mathcal{A}}$, by rewinding $Adv_{\mathcal{A}}$ and swapping the order of (r, r^T). Afterwards, *Sim* can verify whether both values $\bar{y}_j, \bar{y}_j^T$ received from $Adv_{\mathcal{T}}$ are correct. If $Adv_{\mathcal{T}}$ tried to cheat (e.g., if the check of $\bar{y}_j^T$ failed), *Sim* catches $\mathcal{T}$ in doing so and issues the `cheat` instruction. *Sim* aborts in this case (losing any correctness guarantee in case the cheat is not announced). Otherwise, *Sim* continues to play as honest $\mathcal{B}$ and extracts $\mathcal{A}$'s inputs from $\bar{X}$ using K. Note that *Sim* simulates the ideal view of a covert $\mathcal{A}$ with deterrence factor $\epsilon = 1/2$, because for any run in which *Sim* does not receive both keys, $\mathcal{B}$ would detect cheating with probability 1/2 in the actual protocol, in which case it too aborts.

$\mathcal{A}$ *and* $\mathcal{B}$ *trust the token.* We now prove that when the token $\mathcal{T}$ is trusted, the protocol actually realizes the set intersection functionality (i.e., both input privacy in the simulation-based sense and output correctness are achieved). The proof follows closely the one of Theorem 1, indeed since $\mathcal{T}$ is honest, both $\mathcal{A}$'s and $\mathcal{B}$'s input can be extracted by receiving the queries to $\mathcal{T}$, moreover there is no issue of correctness since $\mathcal{T}$ never deviates from the protocol. The only issue to mention is that a malicious $\mathcal{A}$ could play a wrong third message, sending a wrong K^T. Therefore, the ideal world simulator $\mathcal{A}_I$ will first check that $\mathcal{A}$'s message is well formed playing as honest $\mathcal{B}$, and only in case honest $\mathcal{B}$ would have obtained the output, $\mathcal{A}_I$ forwards the extracted input to the ideal functionality. □

Efficiency and Amplifying Deterrence Factor. Overall, the protocol in Fig. 3 approximately doubles the computation performed by $\mathcal{T}$ and the communication between $\mathcal{B}$ and $\mathcal{T}$ compared to the protocol in Fig. 2. The hardware requirements for the token are the same.

In analogy to [30], the deterrence factor ϵ can be increased by using n test elements r_i^T for which $\mathcal{B}$ obtains the corresponding test keys K_i^T from $\mathcal{A}$. Now, $\mathcal{T}$ can only successfully guess the key on which to cheat with probability $p = \frac{1}{n+1}$ s.t. $\epsilon = 1 - p$ is polynomially close to 1 in n. Obviously this is a tradeoff between deterrence factor and efficiency.

6 Only One Token Trusted: Privacy of $\mathcal{A}$'s Input

Model. In this section we extend the model of §4 so that not only $\mathcal{B}$ does not trust the tokens issued by $\mathcal{A}$, but also $\mathcal{B}$ is allowed to collude with all but one hardware manufacturer without $\mathcal{A}$ knowing which one. We show how to detect cheating in this model.

Problem 3 ($\mathcal{B}$ breaks into $\mathcal{T}$ to break privacy of $\mathcal{A}$'s inputs). In the protocols so far, a malicious $\mathcal{B}$ who can break into $\mathcal{T}$ (e.g., by a successful attack or by colluding with $\mathcal{M}$ who builds a trapdoor for $\mathcal{B}$ into $\mathcal{T}$) can obtain k and invert F to recover $\mathcal{A}$'s inputs from $\bar{X}$.

Protocol. To address Problem 3, we extend the protocol of Fig. 3 to multiple tokens as shown in Fig. 4: Instead of using one token, $\mathcal{A}$ uses two hardware tokens $\mathcal{T}_1$ and $\mathcal{T}_2$ manufactured by $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. Then, $\mathcal{A}$ embeds into each token $\mathcal{T}_i$ a different random key and runs the protocol using the sequential composition $F_{K'} = F_{K_2} \circ F_{K_1}$ instead of F_K, i.e., $\mathcal{B}$ communicates first with $\mathcal{T}_1$ and afterwards with $\mathcal{T}_2$. As long as at least one token is resistant against $\mathcal{B}$'s attacks, $\mathcal{B}$ cannot invert $F_{K'}$ and hence cannot recover $\mathcal{A}$'s inputs.

Theorem 3. *If F is a SPRP and f is a PRF, then the protocol depicted in Fig. 4:*

1. *securely evaluates $\mathcal{F}_\cap(X, Y)$ w.r.t. a malicious $\mathcal{B}$ that cannot break into all but one token $\mathcal{T}_i$;*
2. *securely evaluates $\mathcal{F}_\cap(X, Y)$ w.r.t. a covert $\mathcal{A}$ with deterrence factor $\epsilon = 1/2$;*
3. *securely evaluates $\mathcal{F}_\cap(X, Y)$ when both parties can trust all tokens.*

Proof (Sketch). The proof of Theorem 3 follows similarly to that of Theorem 2, but using multiple tokens where $\mathcal{B}$ can break into all but one.

Malicious $\mathcal{B}$ that can break into all but one token $\mathcal{T}_i$. Assume that $\mathcal{B}$ corrupts token $\mathcal{T}_1$ and thus learns k_1, s_1, and s_1^T. Then security for $\mathcal{A}$ follows as in the proof of Theorem 2 from the trustworthiness of $\mathcal{T}_2$, only that we consider the injectively transformed inputs through $F_{k_1}(\cdot)$. Analogously, if $\mathcal{B}$ corrupts $\mathcal{T}_2$ then security follows as before, because the outer function is easy to simulate.

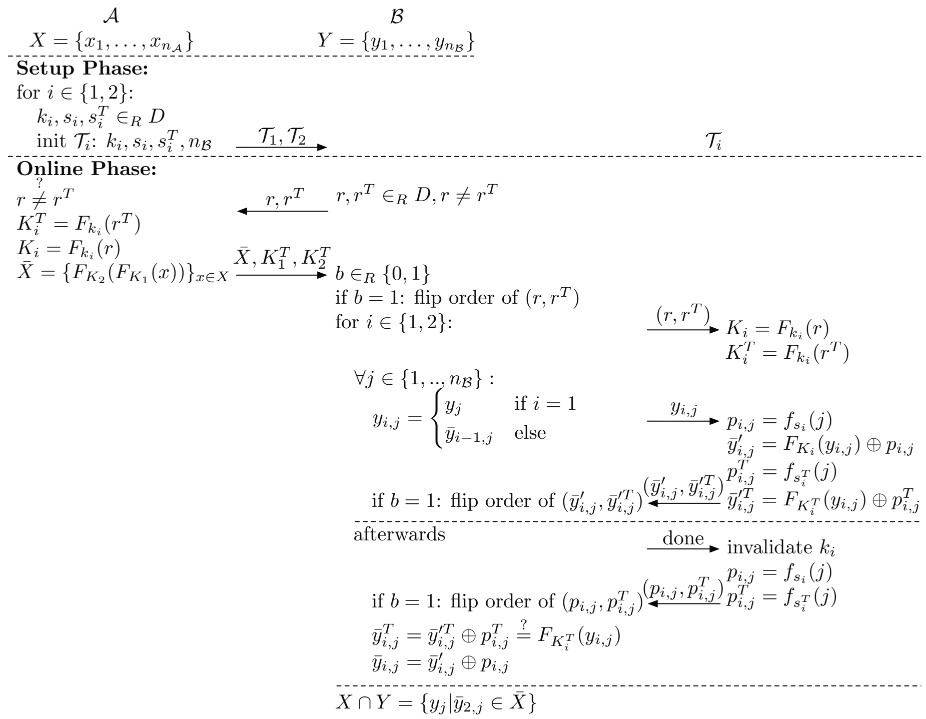

$\mathcal{A}$
$X = \{x_1, \ldots, x_{n_\mathcal{A}}\}$
$\mathcal{B}$
$Y = \{y_1, \ldots, y_{n_\mathcal{B}}\}$
Setup Phase:
for $i \in \{1,2\}$:
$k_i, s_i, s_i^T \in_R D$
init $\mathcal{T}_i$: $k_i, s_i, s_i^T, n_\mathcal{B}$
$\mathcal{T}_1, \mathcal{T}_2$
$\mathcal{T}_i$
Online Phase:
$r \overset{?}{\neq} r^T$
r, r^T
$r, r^T \in_R D, r \neq r^T$
$K_i^T = F_{k_i}(r^T)$
$K_i = F_{k_i}(r)$
$\bar{X} = \{F_{K_2}(F_{K_1}(x))\}_{x \in X}$
$\bar{X}, K_1^T, K_2^T$
$b \in_R \{0,1\}$
if $b = 1$: flip order of (r, r^T)
for $i \in \{1,2\}$:
(r, r^T)
$K_i = F_{k_i}(r)$
$K_i^T = F_{k_i}(r^T)$
$\forall j \in \{1,..,n_\mathcal{B}\}$:
$y_{i,j} = \begin{cases} y_j & \text{if } i = 1 \\ \bar{y}_{i-1,j} & \text{else} \end{cases}$
$y_{i,j}$
$p_{i,j} = f_{s_i}(j)$
$\bar{y}'_{i,j} = F_{K_i}(y_{i,j}) \oplus p_{i,j}$
$p_{i,j}^T = f_{s_i^T}(j)$
$\bar{y}'^T_{i,j} = F_{K_i^T}(y_{i,j}) \oplus p_{i,j}^T$
$(\bar{y}'_{i,j}, \bar{y}'^T_{i,j})$
if $b = 1$: flip order of $(\bar{y}'_{i,j}, \bar{y}'^T_{i,j})$
afterwards
done
invalidate k_i
$p_{i,j} = f_{s_i}(j)$
$p_{i,j}^T = f_{s_i^T}(j)$
$(p_{i,j}, p_{i,j}^T)$
if $b = 1$: flip order of $(p_{i,j}, p_{i,j}^T)$
$\bar{y}_{i,j}^T = \bar{y}'^T_{i,j} \oplus p_{i,j}^T \overset{?}{=} F_{K_i^T}(y_{i,j})$
$\bar{y}_{i,j} = \bar{y}'_{i,j} \oplus p_{i,j}$
$X \cap Y = \{y_j | \bar{y}_{2,j} \in \bar{X}\}$

Fig. 4. Set Intersection Protocol with Privacy of $\mathcal{B}$'s Inputs, (Covert) Correctness of $\mathcal{B}$'s Output and Privacy of $\mathcal{A}$'s Inputs when $\mathcal{A}$ trusts at least one Token

Covert $\mathcal{A}$. The only message $\mathcal{A}$ obtains from $\mathcal{B}$ are the random values r, r^T which do not depend on $\mathcal{B}$'s inputs, and this proves $\mathcal{B}$'s input privacy. For correctness of $\mathcal{B}$'s output, we observe that only one token can cheat while the other behaves correctly such that the probability of being caught remains 1/2. Alternatively, the two tokens could run a combined cheating strategy: token $\mathcal{T}_1$ which is queried first can only guess on which of the two values to cheat without being detected with probability 1/2. In case cheating is not detected, $\mathcal{T}_1$ can transfer information on which value it cheated successfully to $\mathcal{T}_2$ in the value $\bar{y}_{1,j}$. However, the combined cheating strategy will still be caught with probability at least 1/2.

$\mathcal{A}$ and $\mathcal{B}$ trust all tokens. In this case the protocol realizes the set intersection functionalities (i.e., both input privacy in the simulation-based sense and output correctness are achieved). The proof is similar to that of Theorem 2. □

Multiple Tokens and Efficiency. The protocol in Fig. 4 can be generalized to $n \geq 1$ tokens $\mathcal{T}_1, \ldots, \mathcal{T}_n$ manufactured by $\mathcal{M}_1, \ldots, \mathcal{M}_n$, where a malicious $\mathcal{B}$ is able to break all but one token. For $n = 1$, the protocol is equivalent to the protocol of Fig. 3, where $\mathcal{B}$ cannot break into the single token. With n tokens, the protocol in Fig. 4 is essentially a n-times repetition of the protocol in Fig. 3.

Acknowledgments. The work described in this paper was supported in part by the European Commission through the ICT program under contract 238811 UNIQUE, 216676 ECRYPT II, 216499 CACE, and 215270 FRONTS. The first author was supported by an Emmy Noether Grant Fi 940/2-1 of the German Research Foundation (DFG) and by CASED (`www.cased.de`). The second author was supported by the SFEROT project of the European Research Council. The last author was also supported in part by the MIUR Project PRIN "PEPPER: Privacy E Protezione di dati PERsonali" (prot. 2008SY2PH4).

References

1. Hazay, C., Lindell, Y.: Constructions of truly practical secure protocols using standard smartcards. In: CCS 2008, pp. 491–500. ACM, New York (2008)
2. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS 2001, pp. 136–145 (2001)
3. Sharangpani, H.P., Barton, M.L.: Statistical analysis of floating point flaw in the Pentium^{TM}processor. White paper, Intel Corporation (1994)
4. Biham, E., Carmeli, Y., Shamir, A.: Bug attacks. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 221–240. Springer, Heidelberg (2008)
5. Security, H.: Hacker extracts crypto key from TPM chip (2010), `http://www.h-online.com/security/news/item/Hacker-extracts-crypto-key-from-TPM-chip-927077.html`
6. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
7. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
8. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009)
9. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear computational and bandwidth complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010)
10. Ateniese, G., De Cristofaro, E., Tsudik, G.: (If) size matters: Size-hiding private set intersection. Cryptology ePrint Archive, Report 2010/220 (2010), `http://eprint.iacr.org/`
11. Sang, Y., Shen, H.: Privacy preserving set intersection protocol secure against malicious behaviors. In: PDCAT 2007, pp. 461–468. IEEE Computer Society, Los Alamitos (2007)
12. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008)
13. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 125–142. Springer, Heidelberg (2009)
14. Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 418–435. Springer, Heidelberg (2010)

15. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 312–331. Springer, Heidelberg (2010)
16. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. JoC 23, 422–456 (2010)
17. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010)
18. Tate, S., Vishwanathan, R.: Improving cut-and-choose in verifiable encryption and fair exchange protocols using trusted computing technology. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security XXIII. LNCS, vol. 5645, pp. 252–267. Springer, Heidelberg (2009)
19. Fort, M., Freiling, F.C., Penso, L.D., Benenson, Z., Kesdogan, D.: Trustedpals: Secure multiparty computation implemented with smart cards. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 34–48. Springer, Heidelberg (2006)
20. Iliev, A., Smith, S.: More efficient secure function evaluation using tiny trusted third parties. Technical Report TR2005-551, Dartmouth College, Computer Science, Hanover, NH (2005)
21. Hofheinz, D., Müller-Quade, J., Unruh, D.: Universally composable zero-knowledge arguments and commitments from signature cards. In: MoraviaCrypt 2005 (2005)
22. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
23. Järvinen, K., Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 383–397. Springer, Heidelberg (2010)
24. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
25. Moran, T., Segev, G.: David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 527–544. Springer, Heidelberg (2008)
26. Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg (2008)
27. Damgård, I., Nielsen, J.B., Wichs, D.: Universally composable multiparty computation with partially isolated parties. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 315–331. Springer, Heidelberg (2009)
28. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg (2010)
29. Järvinen, K., Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Embedded SFE: Offloading server and network using hardware tokens. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 207–221. Springer, Heidelberg (2010)
30. Kolesnikov, V.: Truly efficient string oblivious transfer using resettable tamper-proof tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 327–342. Springer, Heidelberg (2010)
31. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 137–156. Springer, Heidelberg (2007)