Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries*

Carmit Hazay and Yehuda Lindell

Department of Computer Science
Bar-Ilan University, Israel
{harelc,lindell}@cs.biu.ac.il

Abstract. In this paper we construct efficient secure protocols for *set intersection* and *pattern matching*. Our protocols for securely computing the set intersection functionality are based on secure pseudorandom function evaluations, in contrast to previous protocols that used secure polynomial evaluation. In addition to the above, we also use secure pseudorandom function evaluation in order to achieve secure pattern matching. In this case, we utilize specific properties of the Naor-Reingold pseudorandom function in order to achieve high efficiency.
Our results are presented in two adversary models. Our protocol for secure pattern matching and one of our protocols for set intersection achieve security against *malicious adversaries* under a relaxed definition where one corruption case is simulatable and for the other only privacy (formalized through indistinguishability) is guaranteed. We also present a protocol for set intersection that is fully simulatable in the model of covert adversaries. Loosely speaking, this means that a malicious adversary can cheat, but will then be caught with good probability.

1 Introduction

In the setting of secure two-party computation, two parties wish to jointly compute some function of their private inputs while preserving a number of security properties. In particular, the parties wish to ensure that nothing is revealed beyond the output (privacy), that the output is computed according to the specified function (correctness) and more. The standard definition today (cf. [5] following [13, 4, 17]) formalizes security by comparing a real protocol execution to an "ideal execution" where an incorruptible trusted party helps the parties compute the function. Specifically, in the ideal world the parties just send their inputs (over perfectly secure communication lines) to the trusted party, who computes the function honestly and sends the output to the parties. A real protocol (in which parties interact arbitrarily) is said to be secure if any adversarial attack on a real protocol can essentially be carried out also in the ideal world (of course,

* This research was supported by an Eshkol scholarship and Infrastructures grant from the Israel Ministry of Science and Technology.

in the ideal world the adversary can do almost nothing and this guarantees that the same is true also in the real world). This definition of security is often called *simulation-based* because security is demonstrated by showing that a real protocol execution can be "simulated" in the ideal world.

This setting has been widely studied, and it has been shown that any efficient two-party functionality can be securely computed [24, 12, 11]. These feasibility results demonstrate the wide applicability of secure computation, in principle. However, they fall short of what is needed in implementations because they are far from efficient enough to be used in practice (with a few exceptions). This is not surprising because the results are general and do not utilize any special properties of the specific problem being solved. The focus of this paper is the development of efficient protocols for specific problems of interest.

Relaxed notions of security. Recently, the field of data mining has shown great interest in secure computation, for the purpose of "privacy-preserving data mining". However, most of the protocols that have been constructed with this aim in mind are only secure in the presence of *semi-honest adversaries* who follow the protocol specification (but may try to examine the messages they receive to learn more than they should). Unfortunately, in many cases, this level of security is not sufficient. Rather, adversarial parties are willing to behave maliciously – meaning that they may divert arbitrarily from the protocol specification – in their aim to cheat. It seems that it is hard to obtain highly efficient protocols that are secure in the presence of malicious adversaries (under the standard simulation-based definitions), and two decades after the foundational feasibility results of [12] we only know of very few non-trivial secure computation problems that can be solved with high efficiency in this model. In this paper, we consider two different relaxations in order to achieve higher efficiency:

- *One-sided simulatability:* According to this notion of security, full simulation is provided for one of the corruption cases, while only privacy (via computational indistinguishability) is guaranteed for the other corruption case. This notion of security is useful when considering functionalities for which only one party receives output. In this case, privacy is guaranteed when the party not receiving output is corrupted (and this is formalized by saying that the party cannot distinguish between different inputs used by the other party), whereas full simulation via the ideal/real paradigm is guaranteed when the party receiving output is corrupted. This notion of security has been considered in the past; see [19, 8] for example.
- *Security in the presence of covert adversaries:* This notion of security provides the following guarantee. A malicious adversary may be able to cheat (e.g., learn the other party's private input). However, if it follows such a strategy, it is guaranteed to be caught with probability at least ϵ, where ϵ is called the "deterrence factor" (in this paper, we use $\epsilon = 1/2$). This definition is formalized within the ideal/real simulation paradigm and so has all the advantages offered by it. This definition was recently introduced in [2].

We stress that both notions are relaxations and are not necessarily sufficient for all applications. For example, security in the presence of covert adversaries would not suffice when the computation relates to highly sensitive data or when there are no repercussions to a party being caught cheating. Likewise, the guarantee of privacy alone (as in one-sided simulatability for one of the corruption cases) is sometimes not sufficient. For example, the properties of independence of inputs and correctness are not achieved, and they are sometimes needed. Nevertheless, in many cases, such relaxations are acceptable. Furthermore, using these relaxations, we are able to construct protocols that are much more efficient than anything known that achieves full security in the presence of malicious adversaries (where security is formalized via the ideal/real simulation paradigm).

Secure set intersection. The bulk of this paper is focused on solving the set intersection problem. In this problem, two parties with private sets wish to learn the intersection of their sets and nothing more. There are many cases where such a computation is useful. For example, two health insurance companies may wish to ensure that no one has taken out the same insurance with both of them (if this is forbidden), or the government may wish to ensure that no one receiving social welfare is currently employed and paying income tax. By running secure protocols for these tasks, sensitive information about law-abiding citizens is not unnecessarily compromised.

We present two protocols for this task. The first achieves security in the presence of malicious adversaries with *one-sided simulatability* while the second is secure in the presence of *covert adversaries*. Both protocols take a novel approach. Specifically, instead of using protocols for secure polynomial evaluation [18], our protocols are based on running secure subprotocols for pseudorandom function evaluation. In addition, we use only standard assumptions (e.g., the decisional Diffie-Hellman assumption) and do not resort to random oracles.

In order to get a feel of how our protocol works we sketch the general idea underlying it. The parties run many executions of a protocol for securely computing a pseudorandom function, where one party inputs the key to the pseudorandom function and the other inputs the elements of its set. Denoting the pseudorandom function by F, the input of party P_1 by X and the input of party P_2 by Y, we have that at the end of this stage party P_2 holds the set $\{F_k(y)\}_{y\in Y}$ while P_1 has learned nothing. Then, P_1 just needs to locally compute the set $\{F_k(x)\}_{x\in X}$ and send it to P_2. By comparing which elements appear in both sets, P_2 can learn the intersection (but nothing more). This is a completely different approach to that taken until now that has defined polynomials based on the sets and used secure polynomial evaluations to learn the intersection. We stress that the "polynomial approach" has only been used successfully to achieve security in the presence of semi-honest adversaries [14, 9], or together with random oracles when malicious adversaries are considered [9]. (We exclude the use of techniques that use general zero-knowledge proofs because these are not efficient.)

Secure pattern matching. We present an efficient secure protocol for solving the basic problem of *pattern matching* [3, 15]. In this problem, one party holds a text T and the other a pattern p. The aim is for the party holding the pattern to learn all the locations of the pattern in the text (and there may be many) while the other learns nothing about the pattern. As with our protocols for secure set intersection, the use of secure pseudorandom function evaluation lies at the heart of our solution. However, here we also utilize specific properties of the Naor-Reingold pseudorandom function [20], enabling us to obtain a simple protocol that is significantly more efficient than that obtained by running known general protocols. Our protocol is secure in the presence of *malicious adversaries with one-sided simulatability*, and is the first to address this specific problem.

Related work. The problem of secure set intersection was studied in [9] who presented protocols for both the semi-honest and malicious cases. However, their protocol for the case of malicious adversaries assumes a random oracle. This problem was also studied in [14] whose main focus was the semi-honest model; their protocols for the malicious case use multiple zero-knowledge proofs for proving correct behavior and as such are not very efficient. As we have mentioned, both of the above works use oblivious polynomial evaluation as the basic building block in their solutions.

2 Definitions and Tools

2.1 Definitions

We denote the security parameter by n and computational indistinguishability of ensembles X and Y by $X \stackrel{c}{\equiv} Y$; see [11] for formal definitions. We adopt the convention whereby a machine is said to run in polynomial-time if its number of steps is polynomial in its *security parameter* alone. We use the shorthand PPT to denote probabilistic polynomial-time. Two basic building blocks that we utilize in our constructions are ensembles of pseudorandom functions, denoted by F_{PRF}, and ensembles of pseudorandom permutations, denoted by F_{PRP}, as defined in [10]. We also denote the ensemble of truly random functions by H_{Func} and the ensemble of truly random permutations by H_{Perm}.

One sided simulation for two-party protocols. Two of our protocols achieve a level of security that we call one-sided simulation. In these protocols, P_2 receives output while P_1 should learn nothing. In one-sided simulation, *full simulation* is possible when P_2 is corrupted. However, when P_1 is corrupted we only guarantee *privacy*, meaning that it learns nothing whatsoever about P_2's input (this is straightforward to formalize because P_1 receives no output). This is a relaxed level of security and does not achieve everything we want; for example, independence of inputs and correctness are not guaranteed. Nevertheless, for this level of security we are able to construct highly efficient protocols that are secure in the presence of malicious adversaries. The formal definition appears in the full version; we present it very briefly here. Let $\text{REAL}_{\pi,\mathcal{A}(z),i}(x,y,n)$ denote the output of the honest party and the adversary $\mathcal{A}$ (controlling party P_i) after a real execution of protocol π, where P_1 has input x, P_2 has input y, $\mathcal{A}$ has auxiliary input z, and the security parameter is n. Let $\text{IDEAL}_{f,\mathcal{S}(z),i}(x,y,n)$ be the analogous distribution in an ideal execution with a trusted party who computes f for the parties. Finally, let $\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),i}(x,y,n)$ denote the view of the adversary after a real execution of π as above. Then, we have the following definition:

Definition 1 *Let f be a two-party functionality where only P_2 receives output. We say that a protocol π* securely computes f with one-sided simulation *if the following holds:*

1. *For every non-uniform* PPT *adversary $\mathcal{A}$ in the real model, there exists a non-uniform* PPT *adversary $\mathcal{S}$ for the ideal model, such that for every $x, y, z \in \{0,1\}^*$*
$$\left\{\text{REAL}_{\pi,\mathcal{A}(z),2}(x,y,n)\right\}_{n\in N} \stackrel{c}{\equiv} \left\{\text{IDEAL}_{f,\mathcal{S}(z),2}(x,y,n)\right\}_{n\in N}$$
2. *For every non-uniform* PPT *adversary $\mathcal{A}$, all pairs of inputs $y, y' \in \{0,1\}^*$ with $|y| = |y'|$, and all inputs $x, z \in \{0,1\}^*$,*
$$\left\{\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),1}(x,y,n)\right\}_{n\in N} \stackrel{c}{\equiv} \left\{\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),1}(x,y',n)\right\}_{n\in N}$$

Security in the presence of covert adversaries. In this setting, the adversary may deviate from the protocol specification in an attempt to cheat, and as such is malicious. However, if it follows a strategy which enables it to achieve something that is not possible in the ideal model (like learning the honest party's input), then its cheating is guaranteed to be detected by the honest party with probability at least ϵ, where ϵ is a deterrent parameter. This definition is formalized in three ways in [2]; we consider their strongest definition here. In this definition, the ideal model is modified so that the adversary may send a special cheat message to the trusted party. In such a case, the trusted party tosses coins so that with probability ϵ the adversary is caught and a message corrupted is sent to the honest party (indicating that the other party attempted to cheat). However, with probability $1-\epsilon$, the ideal-model adversary is allowed to cheat and so the trusted party sends it the honest party's full input and also allows it to set the output of the honest party. We refer the reader to [2] and the full version of this paper for further details. The output distribution of an execution of this modified ideal model for a given ϵ and parameters as above is denoted $\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),i}(x,y,n)$. We have the following:

Definition 2 *Let f, π and ϵ be as above. Protocol π is said to* securely compute f in the presence of covert adversaries with ϵ-deterrent *if for every non-uniform* PPT *adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model such that for every $i \in \{1,2\}$, every $x, y \in \{0,1\}^*$ with $|x| = |y|$, and every auxiliary input $z \in \{0,1\}^*$:*

$$\left\{\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),i}(x,y,n)\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),i}(x,y,n)\right\}_{n\in\mathbb{N}}$$

The two notions of security. We remark that one-sided simulatability and security in the presence of covert adversaries are incomparable notions. On the one hand, the guarantees provided by security under one-sided simulation cannot be breached, even by a malicious adversary. This is not the case for security in the presence of covert adversaries where it is possible for a malicious adversary to successfully cheat. On the other hand, the formalization of security for covert adversaries is such that any deviation from what can be achieved in the ideal model is considered cheating (and so will result in the adversary being caught with probability ϵ). This is not the case for one-sided simulatability where one of the parties can make its input depend on the other, or cause the result to not be correctly computed, without ever being caught.

2.2 Tools

In this section, we describe the basic tools used in our constructions. Full descriptions and proofs are provided in the full version of this paper.

Oblivious transfer. We use oblivious transfer in order to achieve secure pseudorandom function evaluation (see below), which in turn is used for our set intersection protocols. For our protocols that achieve one-sided simulatability, we need an oblivious transfer protocol that achieves one-sided simulatability. Such a protocol can be constructed using homomorphic encryption, based on the protocol of [1]. The protocol needs some modifications in order to obtain simulatability in the case that the receiver is corrupted. We can instantiate our protocol with either the El-Gamal [6] or Paillier [21] homomorphic encryptions schemes. However, our instantiation using El-Gamal is considerably more efficient; see the full version. We remark that our protocols actually need to run multiple oblivious transfers in parallel. For the sake of this, we define the *multi-oblivious transfer functionality* with m executions, denoted $\mathcal{F}_{\text{OT}}^{m}$ as follows:

$$((x_1^0, x_1^1), \ldots, (x_m^0, x_m^1), (\sigma_1, \ldots, \sigma_m)) \rightarrow (\lambda, (x_1^{\sigma_1}, \ldots, x_m^{\sigma_m}))$$

Our protocol for computing this functionality works by running the basic protocol in parallel, using the same homomorphic encryption key in each execution. This yields higher efficiency and the number of asymmetric operations per transfer is essentially two. We denote a protocol that securely realizes $\mathcal{F}_{\text{OT}}^{m}$ with one-sided simulation by π_{OT}^{m}.

Our protocol that achieves security for covert adversaries needs an oblivious transfer protocol that is secure for covert adversaries. Such a protocol was presented in [2] and essentially requires 4 exponentiations only per execution.

Oblivious pseudorandom function evaluation. Let $(I_{\text{PRF}}, F_{\text{PRF}})$ be an ensemble of pseudorandom functions, where I_{PRF} is a probabilistic polynomial-time algorithm that generates keys (or more exactly, that samples a function from the ensemble). The task of oblivious pseudorandom function evaluation with F_{PRF} is that of securely computing the functionality $\mathcal{F}_{\text{PRF}}$ defined by

$$(k, x) \mapsto (\lambda, F_{\text{PRF}}(k, x)) \tag{1}$$

where $k \leftarrow I_{\text{PRF}}(1^n)$ and $x \in \{0,1\}^n$.[1] We will use the Naor-Reingold [20] pseudorandom function ensemble F_{PRF} (with some minor modifications). For every n,

[1] If k is not a "valid" key in the range of $I_{\text{PRF}}(1^n)$, then we allow the function to take any arbitrary value. This simplifies our presentation.

the function's key is the tuple $k = (p, q, g^{a_0}, a_1, \ldots, a_n)$, where p is a prime, q is an n-bit prime divisor of $p-1$, $g \in \mathsf{Z}_p^*$ is of order q, and $a_0, a_1, \ldots, a_n \in_R \mathsf{Z}_q^*$. (This is slightly different from the description in [20] but makes no difference to the pseudorandomness of the ensemble.) The function itself is defined by

$$F_{\text{PRF}}(k, x) = g^{a_0 \cdot \prod_{i=1}^{n} a_i^{x_i}} \bmod p$$

We remark that this function is not pseudorandom in the classic sense of it being indistinguishable from a random function whose range is composed of all strings of a given length. Rather, it is indistinguishable from a random function whose range is the group generated by g as defined above. This suffices for our purposes. A protocol for oblivious pseudorandom function evaluation of this function was presented in [8] and involves the parties running an oblivious transfer execution for every bit of the input x. In the full version we prove that the protocol of [8] preserves the security level of the oblivious transfer used (whether it be full security, one-sided simulatability, or security in the presence of covert adversaries). Using the oblivious transfer of [2] we therefore have that for $x \in \{0,1\}^\ell$, the cost of securely computing $\mathcal{F}_{\text{PRF}}$ in the presence of covert adversaries is essentially 4ℓ exponentiations. We remark that by using a multi-oblivious transfer protocol, we can run many executions of π_{PRF} simultaneously. This is of great importance for efficiency.

3 Secure Set-Intersection

In this section we present our main result. We show how to securely compute the two-party set-intersection functionality $\mathcal{F}_\cap$, where each party enters a *set* of values from some predetermined domain. If the input sets are legal, i.e. they are made up of distinct values, then the functionality sends the intersection of these inputs to P_2 and nothing to P_1. Otherwise P_2 is given $\perp$. Let X and Y denote the respective input sets of P_1 and P_2, and let the domain of elements be $\{0,1\}^{p(n)}$ for some known polynomial $p(n)$. We assume that $p(n) = \omega(\log n)$; this is needed for proving security and can always be achieved by padding the elements if necessary. Functionality $\mathcal{F}_\cap$ is defined by:

$$(X, Y) \mapsto \begin{cases} (\lambda, X \cap Y), & \text{if } X, Y \subseteq \{0,1\}^{p(n)} \text{ and are legal sets} \\ (\lambda, \perp), & \text{otherwise} \end{cases}$$

We present two protocols in this section: the first achieves one-sided simulatability in the presence of malicious adversaries, and the second achieves security in the presence of covert adversaries with deterrent $\epsilon = 1/2$.

3.1 Secure Set Intersection with One-Sided Simulatability

The basic idea behind this protocol was described in the introduction. We therefore proceed directly to the protocol, which uses a subprotocol π_{PRF} that securely computes $\mathcal{F}_{\text{PRF}}$ with one-sided simulatability (functionality $\mathcal{F}_{\text{PRF}}$ was defined in Eq. (1) above).

Protocol π_{INT}

- **Inputs:** The input of P_1 is X where $X \subseteq \{0,1\}^{p(n)}$ contains m_1 items, and the input of P_2 is Y where $Y \subseteq \{0,1\}^{p(n)}$ contains m_2 items.
- **Auxiliary inputs:** Both parties have the security parameter 1^n and the polynomial p bounding the lengths of all elements in X and Y. In addition, P_1 is given m_2 (the size of Y) and P_2 is given m_1 (the size of X).
- **The protocol:**
  1. Party P_1 chooses a key $k \leftarrow I_{\text{PRF}}(1^{p(n)})$ for the pseudorandom function. Then, the parties run m_2 parallel executions of π_{PRF}. P_1 enters the key k chosen above in all of the executions, whereas P_2 enters a different value $y \in Y$ in each execution. The output of P_2 from these executions is the set $U = \{(F_{\text{PRF}}(k,y))\}_{y \in Y}$.
  2. P_1 sends P_2 the set $V = \{F_{\text{PRF}}(k,x)\}_{x \in X}$ in a randomly permuted order, where k is the same key P_1 used in Protocol π_{PRF} in the previous step.
  3. P_2 outputs all y's for which $F_{\text{PRF}}(k,y) \in V$. I.e., for every y let f_y be the output of P_2 from π_{PRF} when it used input y. Then, P_2 outputs the set $\{y \mid f_y \in V\}$.

Theorem 3 *Assume that π_{PRF} securely computes $\mathcal{F}_{\text{PRF}}$ with one-sided simulation. Then π_{INT} securely computes $\mathcal{F}_\cap$ with one-sided simulation.*

Proof Sketch: In the case that P_1 is corrupted we need only show that P_1 learns nothing about P_2's inputs. This follows from the fact that the only messages that P_1 receives are in the executions of π_{PRF} which also reveals nothing about P_2's input to P_1. The formal proof of this follows from a standard hybrid argument.

We now proceed to the case that P_2 is corrupted; here we must present a simulator but can also rely on the fact that the π_{PRF} subprotocol is simulatable. Thus, we can analyze the security of π_{INT} in a hybrid model where a trusted party computes $\mathcal{F}_{\text{PRF}}$ for the parties. In this model, P_1 and P_2 just send their inputs to π_{PRF} to the trusted party. Thus, the simulator $\mathcal{S}$ for $\mathcal{A}$ who controls P_2 receives $\mathcal{A}$'s inputs $y_1, \ldots, y_{m_2}$ to the pseudorandom function evaluations. $\mathcal{S}$ chooses a unique random value z_i for each distinct y_i and hands it to $\mathcal{A}$ as its output in the ith evaluation. $\mathcal{S}$ then sends $y_1, \ldots, y_{m_2}$ to the trusted party computing $\mathcal{F}_\cap$ and receives back a subset of the values (this is the output $X \cap Y$); let t be the number of values in the subset. $\mathcal{S}$ completes $X \cap Y$ with a set of $m_1 - t$ random values of length $p(n)$ each, computes the set V from this set as an honest P_1 would and hands it to $\mathcal{A}$.[2] Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. The proof is completed by proving that the ability to distinguish the simulation from a real execution can be converted into the ability to distinguish the pseudorandom function from random. ■

Efficiency. Note first that since π_{PRF} can be run in parallel and has only a constant number of rounds, protocol π_{INT} also has only a constant number of rounds. Next, the number of exponentiations is $O(m_2 \cdot p(n) + m_1)$. This is due to the fact that each local computation of the Naor-Reingold pseudorandom function can be carried out with just one modular exponentiation and n modular multiplications

[2] Since $p(n)$ is superlogarithmic, the probability that any of the random values sent by $\mathcal{S}$ are in P_1's input set is negligible.

(which are equivalent to another exponentiation). Thus, computing the set V requires $O(m_1)$ exponentiations. In addition, for inputs of length $p(n)$, Protocol π_{PRF} consists of running $p(n)$ oblivious transfers (each requiring $O(1)$ exponentiations). Thus m_2 such executions require $O(m_2 \cdot p(n))$ exponentiations. We remark that since $p(n)$ is the size of the input elements it is typically quite small (e.g., the size of an SSN). If this is not the case, then the input can be hashed to a fixed size using a collision-resistant hash function. Thus, $m_2 \cdot p(n) + m_1$ will typically be much smaller than $m_1 \cdot m_2$. (Recall that we do need to assume that $p(n)$ is large enough so that a randomly chosen string does not intersect with any of the sets except with very small probability. However, this can still be quite small.)

We remark that our protocol is much more efficient than that of [14] (although they achieve full simulatability). This is due to the fact that in their protocol every party P_i is required to execute $O(m_1 \cdot m_2)$ zero-knowledge proofs of knowledge, and a similar number of asymmetric computations. (Many of these proofs can be made efficient but not all. In particular, their protocol is only secure as long as the players prove that they do not send the all-zero polynomial. However, no efficient protocols for proving this are known.)

3.2 Secure Set Intersection in the Presence of Covert Adversaries

In this section we present a protocol for securely computing set-intersection in the presence of covert adversaries. Our protocol is based on the high-level idea demonstrated in protocol π_{INT} (achieving one-sided simulation for malicious adversaries). In order to motivate this protocol, we explain why π_{INT} cannot be simulated in the case that P_1 is corrupted. The problem arises from the fact that P_1 may use different keys in the different evaluations of π_{PRF} and in the computation of V. In such a case, the simulator cannot construct a set of values X that corresponds with P_1's behavior. Another problem that arises is that if P_1 can choose the key k by itself, then it can make it so that for some distinct values y and y' it holds that $F_{\text{PRF}}(k, y) = F_{\text{PRF}}(k, y')$. This enables P_2 to effectively make its set X larger, affecting the size of the intersection. Needless to say, this strategy cannot be carried out in the ideal model. Thus, the main objective of the additional steps in our protocol below is to ensure that P_1 uses the same *randomly chosen* k in *all* of the π_{PRF} evaluations as well as in the construction V. This is achieved in the following ways. First, the parties run two series of executions of the π_{PRF} protocol where in one execution real values are used and in the other dummy values are used. Party P_2 then checks that P_1 used the same key in all of dummy executions. This check is carried out by having P_1 and P_2 generate the randomness that P_1 should use in these subprotocols by coin tossing (where P_1 receives coins and P_2 receives a commitment to those coins). Then, P_1 simply reveals the coins used in the dummy series and P_2 can fully verify its behavior. Second, P_1 and P_2 first apply a pseudorandom permutation to their inputs and then a pseudorandom function. Then, P_1 sends two sets V_0 and V_1, and opens one of them to P_2 in order to prove that it was constructed by applying the pseudorandom function with the *same key* as used in

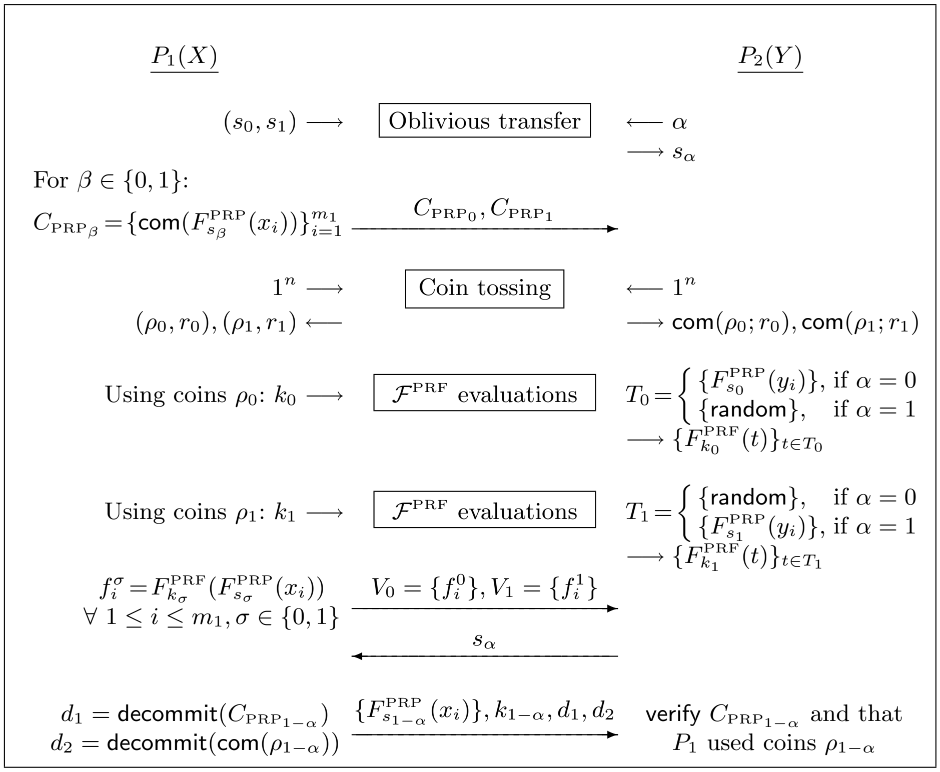

Fig. 1. A high-level diagram of our protocol.

the dummy evaluations. The reason that the pseudorandom permutation is first applied is to hide P_1's values from P_2 when one of the sets V_0, V_1 is "opened". The difficulty in implementing this idea is to devise a way that P_2 can compute the intersection and check all of the above, without revealing more about P_1's input than allowed. Technically, this is achieved by having V_0 equal the set of values $F_{\text{PRF}}(k_0, F_{\text{PRP}}(s_0, x))$ and having V_1 equal the values $F_{\text{PRF}}(k_1, F_{\text{PRP}}(s_1, x))$. Then, P_2 learns either (k_0, s_1) or (k_1, s_0). In this way, it cannot derive any information from the sets (it only knows one of the keys). However, it is enough to check P_1's behavior. A high-level overview of the protocol appears in Figure 1 and the full description (starting with the tools that we use) follows below.

Tools: Our protocol uses the following primitives and subprotocols:

- A pseudorandom permutation with sampling algorithm I_{PRP}. We denote a sampled key by s and the computation of the permutation with key s and input x by $F_{\text{PRP}}(s, x)$.
- A pseudorandom function with sampling algorithm I_{PRF}. We denote a sampled key by k and the computation of the permutation with key k and input x by $F_{\text{PRF}}(k, x)$.
- A perfectly-binding commitment scheme com; we denote by $\mathsf{com}(x; r)$ the commitment to a string x using random coins r.

– An oblivious transfer protocol that is secure in the presence of covert adversaries with deterrent $\epsilon = 1/2$ and can be run in parallel. An efficient protocol that achieves this was presented in [2]. We denote this protocol by π_{OT}.
– An efficient coin-tossing protocol that is secure in the presence of covert adversaries with deterrent $\epsilon = 1/2$. Such a protocol can be constructed by using the protocol of [16], with commitments based on El-Gamal encryption [6] (this enables highly efficient zero-knowledge proofs; see the full version). The exact functionality we need is not plain coin-tossing but rather $(1^n, 1^n) \mapsto ((\rho, r), \mathsf{com}(\rho; r))$ where $\rho \in_R \{0,1\}^n$ and r is random and of sufficient length for committing to ρ. We denote this protocol by π_{CT}.
– A protocol π_{PRF} for computing $\mathcal{F}_{\mathrm{PRF}}$ as defined in Eq. (1), that is secure in the presence of covert adversaries with $\epsilon = 1/2$; see Section 2.2.

We are now ready to present our protocol.

Protocol $\pi_\cap$

– **Inputs:** The input of P_1 is X where $X \subseteq \{0,1\}^{p(n)}$ contains m_1 items, and the input of P_2 is Y where $Y \subseteq \{0,1\}^{p(n)}$ contains m_2 items.
– **Auxiliary inputs:** Both parties have the security parameter 1^n and the polynomial p bounding the lengths of all elements in X and Y. In addition, P_1 is given m_2 (the size of Y) and P_2 is given m_1 (the size of X).
– **The protocol:**
  1. *Oblivious transfer (secure in the presence of covert adversaries):*
     (a) Party P_1 chooses a pair of keys $s_0, s_1 \leftarrow \mathcal{I}_{\mathrm{PRP}}(1^{p(n)})$ for a PRP.
     (b) Party P_2 chooses a random bit $\alpha \in_R \{0,1\}$.
     (c) P_1 and P_2 execute the oblivious transfer protocol π_{OT}. P_1 inputs the keys s_0 and s_1 and plays the sender, and P_2 inputs α and plays the receiver. If one of the parties receives $\mathsf{corrupt}_i$ or abort_i as output, it outputs it and halts. Otherwise P_2 receives s_α.
  2. P_1 computes $C_{\mathrm{PRP}_0} = \{\mathsf{com}(F_{\mathrm{PRP}}(s_0, x))\}_{x \in X}$, $C_{\mathrm{PRP}_1} = \{\mathsf{com}(F_{\mathrm{PRP}}(s_1, x))\}_{x \in X}$ and sends C_{PRP_0} and C_{PRP_1} to P_2.
  3. The parties run the coin-tossing protocol π_{CT} computing $(1^{q(n)}, 1^{q(n)}) \rightarrow ((\rho, r), \mathsf{com}(\rho; r))$ twice, where $q(n)$ is the number of random bits needed to both choose a key $k \leftarrow I_{\mathrm{PRF}}(1^{p(n)})$ and run m_2 executions of the PRF protocol (see below). Party P_1 receives for output (ρ_0, r_0) and (ρ_1, r_1), and P_2 receives $c_{\rho_0} = \mathsf{com}(\rho_0; r_0)$ and $c_{\rho_1} = \mathsf{com}(\rho_1; r_1)$, where ρ_0, ρ_1 are each of length $q(n)$.
  4. *Run oblivious* PRF *evaluations:*
     (a) The parties run m_2 executions of the oblivious PRF evaluation protocol π_{PRF} *in parallel*, in which P_1 inputs the same randomly chosen key $k_0 \leftarrow \mathcal{I}_{\mathrm{PRF}}(1^{p(n)})$ in each execution, and P_2 enters the elements of the set $T_0 = \{F_{\mathrm{PRP}}(s_0, y)\}_{y \in Y}$ (if $\alpha = 0$), and m_2 random values of size $p(n)$ (if $\alpha = 1$). Let U_0 be the set of outputs received by P_2 in these executions. The randomness used by P_1 in all of the executions (and for choosing the key k_0) is the string ρ_0 from the coin-tossing above.
     (b) The parties run another m_2 executions of π_{PRF} *in parallel*, in which P_1 inputs the same randomly chosen key $k_1 \leftarrow \mathcal{I}_{\mathrm{PRF}}(1^{p(n)})$ each time, and P_2 enters m_2 random values of size $p(n)$ (if $\alpha = 0$), and the elements of the set $T_1 = \{F_{\mathrm{PRP}}(s_1, y)\}_{y \in Y}$ (if $\alpha = 1$). Let U_1 be the set of outputs received by P_2 in these executions. The randomness used by P_1 in all of the executions (and for choosing the key k_1) is the string ρ_1 from the coin-tossing above.

5. P_1 computes and sends P_2 the sets of values $V_0 = \{F_{\text{PRF}}(k_0, F_{\text{PRP}}(s_0, x))\}_{x \in X}$ and $V_1 = \{F_{\text{PRF}}(k_1, F_{\text{PRP}}(s_1, x))\}_{x \in X}$, in randomly permuted order.
6. *Run checks:*
   (a) If either $|V_0|$ or $|V_1|$ are smaller than m_1 or not distinct, P_2 outputs $\mathsf{corrupted}_1$, otherwise it sends P_1 the key s_α.
   (b) If P_2 sends s such that $s \notin \{s_0, s_1\}$, then P_1 halts. Otherwise, P_1 sets α such that $s = s_\alpha$. Then, P_1 sends P_2 the decommitments for all values in the set $C_{\text{PRP}_{1-\alpha}}$, and the decommitment of $c_{\rho_{1-\alpha}}$.
   (c) Let $W_{1-\alpha}$ denote the opening of $C_{\text{PRP}_{1-\alpha}}$ and $\rho_{1-\alpha}$ the opening of $c_{\rho_{1-\alpha}}$. First, P_2 checks that the responses of P_1 to its messages in the m_2 executions of the PRF evaluations in which it input random strings are exactly the responses of an honest P_1 using random coins $\rho_{1-\alpha}$ to generate $k_{1-\alpha}$ and run the subprotocols. Furthermore, P_2 checks that $V_{1-\alpha} = \{F_{\text{PRF}}(k_{1-\alpha}, w)\}_{w \in W_{1-\alpha}}$ using $k_{1-\alpha}$ as above. In case the above does not hold, P_2 outputs $\mathsf{corrupted}_1$. Otherwise, let f_y be the output received by P_2 from the PRF evaluation in which it input $F_{\text{PRP}}(s_\alpha, y)$. Party P_2 outputs the set $\{y \mid f_y \in V_\alpha\}$.

We now prove the security of the protocol:

Theorem 4 *Assume that $\pi_{\text{OT}}, \pi_{\text{CT}}, \pi_{\text{PRF}}$ are secure in the presence of covert adversaries with deterrent $\epsilon = \frac{1}{2}$, and assume that* com *is a perfectly-binding commitment scheme and that F_{PRF} and F_{PRP} are pseudorandom function and permutation families, respectively. Then Protocol $\pi_\cap$ securely computes the set-intersection functionality $\mathcal{F}_\cap$ in the presence of covert adversaries with $\epsilon = \frac{1}{2}$.*

Proof: We will separately consider the case that P_1 is corrupted and the case that P_2 is corrupted. The case where both parties are honest is straightforward and therefore omitted. We present the proof in a hybrid model in which a trusted party is used to compute the oblivious transfer and coin-tossing computations. We denote these functionalities by $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{CT}}$. (Unfortunately, we cannot do the same for π_{PRF} because P_1 needs to use the coins ρ_0, ρ_1 in the protocol.)

Party P_1 is corrupted. Let $\mathcal{A}$ be an adversary controlling the party P_1; we construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ receives X and z, and invokes $\mathcal{A}$ on this input.
2. $\mathcal{S}$ plays the trusted party for the oblivious transfer execution with $\mathcal{A}$ as the sender, and receives the input that $\mathcal{A}$ sends to the trusted party:
   (a) If this input is abort_1 or $\mathsf{corrupted}_1$, then $\mathcal{S}$ sends abort_1 or $\mathsf{corrupted}_1$ (respectively) to the trusted party computing $\mathcal{F}_\cap$, simulates P_2 aborting and halts (outputting whatever $\mathcal{A}$ outputs).
   (b) If the input is cheat_1, then $\mathcal{S}$ sends cheat_1 to the trusted party. If it receives back $\mathsf{corrupted}_1$, then it hands $\mathcal{A}$ the message $\mathsf{corrupted}_1$ as if it received it from the trusted party, simulates P_2 aborting and halts (outputting whatever $\mathcal{A}$ outputs). If it receives back $\mathsf{undetected}$ (and the input set Y of the honest P_2) then $\mathcal{S}$ proceeds as follows. First, it hands $\mathcal{A}$ the message $\mathsf{undetected}$ together with a random α that $\mathcal{A}$ expects to receive (as P_2's input to π_{OT}). Next, it uses the input Y of P_2 that it

obtained in order to perfectly emulate P_2 in the rest of the execution. That is, it runs P_2's honest strategy with input Y while interacting with $\mathcal{A}$ playing P_1 for the rest of the execution. Let Z be the output for P_2 that it receives. $\mathcal{S}$ sends Z to the trusted party (for P_2's output) and outputs whatever $\mathcal{A}$ outputs. The simulation ends here in this case.

    (c) If the input is a pair of keys s_0, s_1, $\mathcal{S}$ proceeds with the simulation below.[3]

3. $\mathcal{S}$ receives from $\mathcal{A}$ two sets of commitments C_{PRP_0} and C_{PRP_1}.
4. $\mathcal{S}$ receives from $\mathcal{A}$ its input for $\mathcal{F}_{\text{CT}}$. In case it equals abort_1, $\mathsf{corrupted}_1$, or cheat_1, then $\mathcal{S}$ behaves exactly as above in the OT execution. Otherwise $\mathcal{S}$ chooses random (ρ_0, r_0) and (ρ_1, r_1) of the appropriate length and hands them to $\mathcal{A}$.
5. $\mathcal{S}$ runs the simulator $\mathcal{S}_{\text{PRF}}$ guaranteed to exist for the protocol π_{PRF} (by the assumption that it is secure) on the residual $\mathcal{A}$ at this point (i.e., $\mathcal{S}$ defines an adversary $\mathcal{A}'$ that is just $\mathcal{A}$ with the messages sent until now hardwired into it). If $\mathcal{S}_{\text{PRF}}$ wishes to send abort_1, $\mathsf{corrupted}_1$ or cheat_1 in any of the executions, then $\mathcal{S}$ acts exactly as above. Otherwise, $\mathcal{S}$ proceeds. Let t be the transcript of messages sent by $\mathcal{A}$ in the simulated view of π_{PRF} as generated by $\mathcal{S}_{\text{PRF}}$ (we define the residual $\mathcal{A}$ so that it outputs this transcript and so this is also what is output by $\mathcal{S}_{\text{PRF}}$).
6. $\mathcal{S}$ receives from $\mathcal{A}$ two sets of computed values V_0 and V_1. If they are not of size m_1 or not distinct, $\mathcal{S}$ sends $\mathsf{corrupted}_1$ to the trusted party, simulates P_2 aborting and halts (outputting whatever $\mathcal{A}$ outputs).
7. Otherwise, $\mathcal{S}$ hands $\mathcal{A}$ the key s_0 and receives back $\mathcal{A}$'s decommitments of C_{PRP_1} and c_{ρ_1}. $\mathcal{S}$ then rewinds $\mathcal{A}$, hands it s_1 and receives back its decommitments of C_{PRP_0} and c_{ρ_0}. Simulator $\mathcal{S}$ runs the same checks as an honest P_2 would run (it uses the transcript t to check that $\mathcal{A}$ acted honestly using the randomness ρ_0, ρ_1). We have two cases:
    (a) *Case 1 – all of the checks carried by $\mathcal{S}$ in both rewindings pass:* Let k_0 and k_1 denote the keys that an honest P_1 would have used in the PRF evaluations when its coins are ρ_0 and ρ_1, respectively (where ρ_b is value committed to in c_{ρ_b}). Then, $\mathcal{S}$ chooses a random bit $\alpha \in_R \{0,1\}$ and sends the trusted party the set $\{F^{-1}_{\text{PRP}}(s_\alpha, w)\}_{w \in W_\alpha}$.
    (b) *Case 2 – there exists a bit $\alpha \in \{0,1\}$ so that the checks when $\mathcal{S}$ sent $s_{1-\alpha}$ failed:* Simulator $\mathcal{S}$ sends cheat_1 to the trusted party. If it receives back $\mathsf{corrupted}_1$ then it rewinds $\mathcal{A}$ and sends it $s_{1-\alpha}$ again. If it receives back $\mathsf{undetected}$ then it rewinds $\mathcal{A}$ and sends it s_α. Then, it runs the last step of the protocol exactly as P_2 would, using P_2's real input. $\mathcal{S}$ then sends the trusted party whatever P_2 would output in the ideal model.
8. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

Let $\epsilon = \frac{1}{2}$. We prove that for every $X \subseteq \{0,1\}^{p(n)}$ of size m_1 and $Y \subseteq \{0,1\}^{p(n)}$ of size m_2, and every $z \in \{0,1\}^*$

$$\left\{\text{IDEALSC}^{\epsilon}_{\mathcal{F}_\cap, \mathcal{S}(z), 1}(X, Y, n)\right\}_{n \in N} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}^{\text{OT,CT}}_{\pi_\cap, \mathcal{A}(z), 1}(X, Y, n)\right\}_{n \in N}$$

[3] We assume a mapping from *any* string to a valid key for the pseudorandom permutation.

Recall that in the above $\{\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{CT}}\}$-hybrid model, the view of P_1 includes its output from $\mathcal{F}_{\text{CT}}$, the messages sent during the π_{PRF} executions, and the value s_α that P_2 sends after receiving V_0 and V_1. Thus the only difference between the hybrid and ideal executions is within the π_{PRF} executions. This is due to the fact that $\mathcal{S}$ invokes $\mathcal{S}_{\text{PRF}}$ whereas in a hybrid execution a real π_{PRF} execution is run between P_1 and P_2. Clearly, the views of $\mathcal{A}$ in these executions are computationally indistinguishable. The more interesting challenge is thus to prove that the joint output distributions of P_2 and these views are computationally indistinguishable.

We consider three different cases. In the first case $\mathcal{A}$'s input to $\mathcal{F}_{\text{OT}}$ or $\mathcal{F}_{\text{CT}}$ is either $\mathsf{corrupted}_1$, abort_1 or cheat_1. Let bad_1 denote this event. In this case, the execution is either aborted (with P_2 receiving abort_1 or $\mathsf{corrupted}_1$) or $\mathcal{S}$ receives the honest P_2's full input with which to perfectly complete the simulation. Thus,

$$\{\text{IDEALSC}^{\epsilon}_{\mathcal{F}_\cap,\mathcal{S}(z),1}(X,Y,n) \mid \mathsf{bad}_1\} \equiv \{\text{HYBRID}^{\text{OT,CT}}_{\pi,\mathcal{A}(z),1}(X,Y,n) \mid \mathsf{bad}_1\}$$

In the second case, $\mathcal{A}$ provides valid inputs for $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{CT}}$, yet there exists an $\alpha \in \{0,1\}$ value for which $\mathcal{A}$ does not provide a valid response in Step 6 of the protocol; denote this event by bad_2. Now, if P_2 sent α to $\mathcal{F}_{\text{OT}}$ then $\mathcal{A}$ cannot deviate from the protocol within the π_{PRF} executions on $T_{1-\alpha}$ without *definitely* getting caught by P_2 (and the simulator). Thus, in both the hybrid and ideal executions, P_2 outputs $\mathsf{corrupted}_1$ with the same probability. Furthermore, when it does not output $\mathsf{corrupted}_1$, simulator $\mathcal{S}$ concludes the simulation with P_2's real input (note that although these inputs are already used earlier in π_{PRF}, since $\mathcal{S}$ knows the values k_0, k_1 it can conclude the simulation even when receiving P_2's inputs later). Thus, the only difference is that in the real protocol, the π_{PRF} executions are run with P_2's inputs whereas in the simulation $\mathcal{S}_{\text{PRF}}$ is used. By the security of $\mathcal{S}_{\text{PRF}}$ we have:

$$\{\text{IDEALSC}^{\epsilon}_{\mathcal{F}_\cap,\mathcal{S}(z),1}(X,Y,n) \mid \mathsf{bad}_2\} \stackrel{c}{\equiv} \{\text{HYBRID}^{\text{OT,CT}}_{\pi,\mathcal{A}(z),1}(X,Y,n) \mid \mathsf{bad}_2\}$$

The last case we need to consider is when neither bad_1 nor bad_2 occur; denote this event by $\neg\mathsf{bad}$. Let k_0 and k_1 be the keys that $\mathcal{A}$ used in all of the π_{PRF} executions, and let s_0 and s_1 be the values that $\mathcal{A}$ input to the oblivious transfer. Then we have the following claim:

Claim 5 *Let $X_\alpha = \{F^{-1}_{\text{PRP}}(s_\alpha, w)\}_{w \in W_\alpha}$ and consider the event $\neg\mathsf{bad}$ where neither bad_1 nor bad_2 occur. Then, for every $\alpha \in \{0,1\}$ and set $Y \subseteq \{0,1\}^{p(n)}$, it holds that $z \in X_\alpha \cap Y$ if and only if $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, z)) \in V_\alpha \cap U_\alpha$, except with negligible probability.*

Proof Sketch: If $z \in X_\alpha \cap Y$, then $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, z)) \in V_\alpha \cap U_\alpha$ because $\mathcal{A}$ uses the same key k_α for the PRF evaluation that defines U_α and for computing V_α. If this were not the case, then $\mathcal{A}$ would be caught cheating with probability at least $1/2$ (whereas here we are dealing with the case that $\mathcal{A}$ provides answers that never result in it being caught cheating).

As for the other direction, assume that $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, z)) \in V_\alpha \cap U_\alpha$. Then a problem can arise if there exist $y \in Y$ and $x \in X$ such that $x \neq y$ and yet $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, x)) = F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, y))$. If $\mathcal{A}$ could choose X after k_α is known, then it could indeed cause such an event to happen. However, notice that $\mathcal{A}$ is committed to its inputs (in C_{PRP_0} and C_{PRP_1}) before k_α is chosen in the coin tossing. Thus, the probability that such a "collision" occurs, where the probability is taken over the choice of k_α and the sets X and Y are already fixed, is negligible (or else F_{PRF} can be distinguished from random). ■

This implies that the output received by P_2 in the hybrid and ideal executions is the same (except with negligible probability). Combining this with the fact that the view of $\mathcal{A}$ is clearly indistinguishable in both executions, we have:

$$\{\text{IDEALSC}^{\epsilon}_{\mathcal{F}_\cap,\mathcal{S}(z),1}(X,Y,n) \mid \neg\mathsf{bad}\} \stackrel{c}{\equiv} \{\text{HYBRID}^{\text{OT,CT}}_{\pi,\mathcal{A}(z),1}(X,Y,n) \mid \neg\mathsf{bad}\}$$

Combining the above three cases, and noting that the events bad_1 and bad_2 happen with probability that is negligibly close in the hybrid and ideal executions, we have that the output distributions are computationally indistinguishable, as required.

Party P_2 is corrupted. Let $\mathcal{A}$ be an adversary controlling party P_2. We construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ receives Y and z, and invokes $\mathcal{A}$ on this input.
2. $\mathcal{S}$ plays the trusted party for the oblivious transfer execution with $\mathcal{A}$ as the receiver. $\mathcal{S}$ receives the input that $\mathcal{A}$ sends to the trusted party. If this input is abort_2, $\mathsf{corrupted}_2$ or cheat_2, then $\mathcal{S}$ works in an analogous way as when this occurs in the simulation when P_1 is corrupted.
   If the input equals a bit α, then $\mathcal{S}$ samples a key $s_\alpha \leftarrow \mathcal{I}_{\text{PRP}}(1^{p(n)})$ as the honest P_1 does, and hands it to $\mathcal{A}$ emulating $\mathcal{F}_{\text{OT}}$'s answer. $\mathcal{S}$ samples a second key $s_{1-\alpha} \leftarrow \mathcal{I}_{\text{PRP}}(1^{p(n)})$ as above, and keeps it for later.
3. $\mathcal{S}$ sends $\mathcal{A}$ two sets of m_2 commitments C_{PRP_0} and C_{PRP_1} to distinct random values of length $p(n)$.
4. $\mathcal{S}$ receives from $\mathcal{A}$ its input for $\mathcal{F}_{\text{CT}}$. In case it equals abort_2, $\mathsf{corrupted}_2$, or cheat_2, then $\mathcal{S}$ behaves exactly as above in the OT execution. Otherwise $\mathcal{S}$ chooses random (ρ_0, r_0) and (ρ_1, r_1) of the appropriate length and hands $c_{\rho_0} = \mathsf{com}(\rho_0; r_0)$ and $c_{\rho_1} = \mathsf{com}(\rho_1; r_1)$ to $\mathcal{A}$.
5. $\mathcal{S}$ simulates the PRF evaluations as follows. If $\alpha = 0$ (where α is $\mathcal{A}$'s input to the oblivious transfer), then $\mathcal{S}$ runs the simulator $\mathcal{S}_{\text{PRF}}$ on the residual $\mathcal{A}$ for the first m_2 executions, and follows the honest P_1's instructions using random coins ρ_1 for the second m_2 executions (where the "first" and "second" set is as in the order described in the protocol). In contrast, if $\alpha = 1$, then $\mathcal{S}$ follows the honest P_1's instructions using random coins ρ_0 for the first m_2 executions and runs the simulator $\mathcal{S}_{\text{PRF}}$ on the residual $\mathcal{A}$ for the second m_2 executions.
   In the m_2 executions simulated by $\mathcal{S}_{\text{PRF}}$, simulator $\mathcal{S}$ receives the input that $\mathcal{S}_{\text{PRF}}$ wishes to send to the trusted party as its input in the PRF executions:

(a) If any of these inputs is abort_2, $\mathsf{corrupted}_2$, or cheat_2, then $\mathcal{S}$ behaves exactly as above in the OT execution.

(b) Else, let T' denote the set of m_2 elements (with length bounded by $p(n)$) that $\mathcal{S}_{\text{PRF}}$ wishes to send as $\mathcal{A}$'s inputs to π_{PRF}. Then $\mathcal{S}$ hands $\mathcal{S}_{\text{PRF}}$ the set $\{F_{\text{PRF}}(k_\alpha, t)\}_{t \in T'}$ as its output from the trusted party, where $k_\alpha \leftarrow I_{\text{PRF}}(1^{p(n)})$ is a randomly generated key. In addition, $\mathcal{S}$ defines the set $Y' = \{F^{-1}_{\text{PRP}}(s_\alpha, t)\}_{t \in T'}$. (If Y' is not exactly of size m_2, then $\mathcal{S}$ adds $m_2 - |Y'|$ random elements of size $p(n)$; recall that $p(n) = \omega(\log n)$ and so random values are in the intersection with only negligible probability.)

6. $\mathcal{S}$ sends the trusted party computing $\mathcal{F}_\cap$ the set Y' that it recorded and receives back for output the set Z (note $Z = X \cap Y'$). Then it chooses $m_2 - |Z|$ distinct random elements and adds them to Z. Finally, $\mathcal{S}$ computes and sends $\mathcal{A}$ the sets $V_\alpha = \{F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, z))\}_{z \in Z}$ and $V_{1-\alpha} = \{F_{\text{PRF}}(k_{1-\alpha}, w)\}_{\mathsf{com}(w) \in C_{\text{PRP}_{1-\alpha}}}$. We remark that the elements of V_α are randomly permuted before being sent.
7. $\mathcal{S}$ receives from $\mathcal{A}$ the value s_α and responds with the decommitments of $C_{\text{PRP}_{1-\alpha}}$ and the decommitment of $c_{\rho_{1-\alpha}}$. If $\mathcal{A}$ did not send s_α, then $\mathcal{S}$ halts.
8. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Let $\epsilon = \frac{1}{2}$. We prove that for every $X \subseteq \{0,1\}^{p(n)}$ of size m_1 and $Y \subseteq \{0,1\}^{p(n)}$ of size m_2, and every $z \in \{0,1\}^*$

$$\left\{\text{IDEALSC}^{\epsilon}_{\mathcal{F}_\cap, \mathcal{S}(z), 2}(X, Y, n)\right\}_{n \in N} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}^{\text{OT,CT}}_{\pi_\cap, \mathcal{A}(z), 2}(X, Y, n)\right\}_{n \in N}$$

Note first that the simulation differs from a real execution with respect to how the sets C_{PRF_α} and V_α are generated, and with respect to the decommitments of $C_{\text{PRF}_{1-\alpha}}$ (recall that in the real execution P_1 uses its input X for these computations whereas the simulator does not know X). Nevertheless, the views cannot be distinguished due to the hiding property of F_{PRF}, F_{PRP} and com. As in the previous analysis, we begin with the case where $\mathcal{A}$ sends abort_2, cheat_2 or $\mathsf{corrupted}_2$ to $\mathcal{F}_{\text{OT}}$ or $\mathcal{F}_{\text{CT}}$. Due to the similarity to the case were P_1 is corrupted we omit the details here. Let bad denote the event where $\mathcal{A}$ sends abort_2, $\mathsf{corrupted}_2$ or cheat_2. Then relying on the above discussion it holds that,

$$\{\text{IDEALSC}^{\epsilon}_{\mathcal{F}_\cap, \mathcal{S}(z), 2}(X, Y, n) \mid \mathsf{bad}\} \equiv \{\text{HYBRID}^{\text{OT,CT}}_{\pi, \mathcal{A}(z), 2}(X, Y, n) \mid \mathsf{bad}\}$$

Next we analyze the security of P_1 in case $\mathcal{A}$ provides valid inputs to $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{CT}}$, and prove through the following series of games that the output distributions are computationally indistinguishable. For lack of space in this abstract, we only sketch this part of the proof.

Game H$_1$**:** In the first game the simulator has access to an oracle $\mathcal{O}_{F_{\text{PRP}}}$ for computing F_{PRP} such that instead of computing F_{PRP} using $s_{1-\alpha}$, it queries the oracle. Clearly the output distribution of the current and original simulation is identical.

Game H$_2$**:** In this game we replace $\mathcal{O}_{F_{\text{PRP}}}$ with an oracle $\mathcal{O}_{H_{\text{Perm}}}$ computing a truly random permutation while the rest of the execution is as above. Indistinguishability holds using a standard reduction.

Game H_3**:** The next game is identical to the previous one except that the simulator knows the real input X of P_1 but uses it only for the computation of $V_{1-\alpha}$ and $C_{\text{PRP}_{1-\alpha}}$. Since the oracle is a truly random permutation, the distribution here is identical (note that X is a set and thus all items are distinct).
Game H_4**:** In this game the simulator is given an oracle $\mathcal{O}_{F_{\text{PRF}}}$ for computing F_{PRF} (with a random key) which it uses instead of computing F_{PRF} using k_α. The only difference is that in H_3, the coins used to generate k_α are committed to in c_{ρ_α} whereas in H_4 the oracle uses a random key that is independent of those coins. The fact that these games are indistinguishable therefore follows from the hiding property of the commitment scheme. Note that the executions using $k_{1-\alpha}$ remain the same.
Game H_5**:** Next we replace $\mathcal{O}_{F_{\text{PRF}}}$ with a truly random function $\mathcal{O}_{H_{\text{Func}}}$; indistinguishability here follows from the pseudorandomness of F_{PRF}.
Game H_6**:** In this game we let the simulator query its PRF oracle on the real input set X of P_1. That is, the simulator uses X for the entire computation as the real party P_1. Now, since $\mathcal{O}_{H_{\text{Func}}}$ is a truly random function, we have the same output distribution in both games.
Game H_7**:** Here we modify $\mathcal{O}_{H_{\text{Perm}}}$ back into $\mathcal{O}_{F_{\text{PRP}}}$. This replacement affects the PRP computation for the $(1-\alpha)$th set of PRP evaluations.
Game H_8**:** In this game we modify $\mathcal{O}_{H_{\text{Func}}}$ back into $\mathcal{O}_{F_{\text{PRF}}}$.
Game H_{10}**:** Finally, we let the simulator conduct the PRF and PRP computations by itself. This does not affect the outputs of these functions, but as above a reduction to the hiding property of the commitment c_{ρ_α} is needed because now the coins used to generate the key k_α are committed to in c_{ρ_α}. In addition we let $\mathcal{S}$ compute C_{PRP_α} as the honest P_1 would. Since $\mathcal{S}$ is not required to decommit these commitments, it is again easy to reduce indistinguishability here to the hiding property of these commitments.

We therefore conclude that H_{10} is computationally indistinguishable from the (original) ideal simulation by $\mathcal{S}$. However, H_{10} is identical to the real execution in the hybrid model, completing the proof. ■

Efficiency. We analyze the complexity of protocol $\pi_\cap$. We first count the number of asymmetric operations; in particular, modular exponentiations. Note that each invocation of π_{PRF} with inputs of length $p(n)$ requires $4p(n)+1$ exponentiations, because every invocation of the covert oblivious transfer requires at most 4 such computations, and π_{PRF} runs an oblivious transfer for every bit of P_2's input (one additional exponentiation is used for obtaining the final result). Given that there are $2m_2$ executions of π_{PRF}, we have that the number of exponentiations is approximately $8m_2 \cdot (p(n)+1) + m_1$. As we have already mentioned, $p(n)$ is expected to be quite small in most cases (and a collision-resistant hash function can be used when not). We note that our protocol is completely modular meaning that any protocol π_{PRF} for any pseudorandom function F_{PRF} can be used. Thus, the development of a more efficient protocol π_{PRF} will automatically result in our protocol also being more efficient. In terms of round efficiency, $\pi_\cap$ has a constant number of rounds due to the round efficiency of π_{OT} in the covert model, and the fact that all these executions are run in parallel.

4 Secure Pattern Matching

The basic problem of *pattern matching* is the following one: given a text T of length N and a pattern p of length m, find all the locations in the text where pattern p appears in the text. Stated differently, for every $i = 1, \ldots, N - m + 1$, let T_i be the substring of length m that begins at the ith position in T. Then, the basic problem of pattern matching is to return the set $\{i \mid T_i = p\}$. This problem has been intensively studied and can be solved optimally in time that is linear in size of the text [3, 15].

In this section, we address the question of how to securely compute the above basic pattern matching functionality. The functionality, denoted $\mathcal{F}_{\text{PM}}$, is defined by

$$((T, m), p) \mapsto \begin{cases} (\lambda, \{i \mid T_i = p\}) & \text{if } |p| \leq m \\ (\lambda, \{i \mid T_i = p_1 \ldots p_m\}) & \text{otherwise} \end{cases}$$

where T_i is defined as above, T and p are binary strings and p_i is the ith bit in p. Note that P_1 who holds the text learns nothing about the pattern held by P_2, and the only thing that P_2 learns about the text held by P_1 is the locations where its pattern appears.

Although similar questions have been considered in the past (e.g., keyword search [8]), to the best of our knowledge, this is the first work considering the basic problem of pattern matching as described above. The main difference between keyword search and the problem that we consider here is that in keyword search, each keyword is assumed to appear only once. However, here the text is viewed as a stream and a pattern can appear multiple times. Furthermore, the strings $T_i, T_{i+1}, \ldots$ are dependent on each other (adjacent T_i's only differ in their first and last characters). Thus, it is not possible to apply a pseudorandom function to each T_i and use a protocol to securely compute $\mathcal{F}_{\text{PRP}}$ on p as in the case of keyword search. Thus it seems that finding a secure solution for this problem is harder.

We present a protocol for securely computing $\mathcal{F}_{\text{PM}}$ in the presence of *malicious adversaries* with *one-sided simulatability*. The basic idea behind our protocol is for P_1 and P_2 to run a *single* execution of π_{PRF} for securely computing a pseudorandom function with one-sided simulatability; let $f = F_{\text{PRF}}(k, p)$ be the output received by P_2. Then, P_1 locally computes the pseudorandom function on T_i for every i and sends the results $\{F_{\text{PRF}}(k, T_i)\}$ to P_2. Party P_2 can then find all the matches by just seeing where f appears in the series sent by P_1. Unfortunately, within itself, this is insufficient because P_2 can then detect repetitions within T. That is, if $T_i = T_j$ then P_2 will learn this because this implies that $F_{\text{PRF}}(k, T_i) = F_{\text{PRF}}(k, T_j)$. However, if $T_i \neq p$, this should not be revealed. We therefore include the index i of the subtext T_i in the computation and have P_1 send the values $F_{\text{PRF}}(k, T_i \| \langle i \rangle)$ where $\langle i \rangle$ denotes the binary representation of i. This in turns generates another problem because now it is not possible for P_2 to see where p appears given only $F_{\text{PRF}}(k, p)$; this is solved by having P_2 obtain $F_{\text{PRF}}(k, p \| \langle i \rangle)$ for every i. Although this means that P_2 obtains n different outputs of F_{PRF} (because there are n different indices i), we utilize specific properties of the Naor-Reingold pseudorandom function, and the protocol π_{PRF} for computing it, in order to have P_2 obtain all of these values while running only a *single* execution of π_{PRF}. Due to lack of space, we defer the description of how this is achieved to the full version.

Protocol π_{PM}

- **Inputs:** The input of P_1 is a binary string T of size N, and the input of P_2 is a binary pattern p of size m.
- **Auxiliary Inputs:** the security parameter 1^n, and the input sizes N and m.
- **The protocol:**
  1. Party P_1 chooses a key for computing the Naor-Reingold function on inputs of length $m + \log N$; denote the key $k = (p, q, g^{a_0}, a_1, \ldots, a_{m+\log N})$.
  2. The parties execute a modified version of π_{PRF} for computing the Naor-Reingold function, where P_1 enters the key k and P_2 enters its pattern p of length m. The modification is such that P_2's output is the *set* $\{f_i = F_{\text{PRF}}(k, p\|\langle i\rangle)\}_{i=1}^{N-m+1}$, rather than just a single value.
  3. For every i, let $t_i = F_{\text{PRF}}(k, T_i\|\langle i\rangle)$. Then, P_1 sends P_2 the set $\{(i, t_i)\}_{i=1}^{N-m+1}$.
  4. P_2 outputs the set of indices $\{i\}$ for which $f_i = t_i$.

Theorem 6 *Let F_{PRF} denote the Naor-Reingold function and assume that it is pseudorandom. Furthermore, assume that protocol π_{PRF} securely computes the functionality $(k, p) \mapsto (\lambda, \{F_{\text{PRF}}(k, p\|\langle i\rangle)\}_{i=1}^{N-m+1})$ in the presence of malicious adversaries with one-sided simulatability. Then protocol π_{PM} securely computes $\mathcal{F}_{\text{PM}}$ in the presences of malicious adversaries with one-sided simulatability.*

Proof Sketch: For the case that P_1 is corrupted, we need to show that P_1 learns nothing about P_2's input. This follows immediately from the fact that π_{PRF} is secure with one-sided simulatability and P_1 receives no other messages. For the case that party P_2 is corrupted we need to present a simulator. Very briefly, the simulator $\mathcal{S}$ works by obtaining the pattern p that $\mathcal{A}$ inputs to π_{PRF} and generating values t_i that are completely random when $p \neq T_i$ and that equal f_i when $p = T_i$ ($\mathcal{S}$ knows when $p \neq T_i$ and when $p = T_i$ because this is given by the output received from the trusted party). The security is thus reduced to the pseudorandomness of the Naor-Reingold function. ■

Efficiency. π_{PM} has a constant number of rounds, and each parties carries out approximately $2N$ exponentiations where N is the length of the text.

References

1. W. Aiello, Y. Ishai, and O. Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. *EUROCRYPT '01*, Springer-Verlag (LNCS 2045), pages 110–135, 2001.
2. Y. Aumann, and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *TCC 2007*, Springer-Verlag (LNCS 4392), pages 137–156, 2007.
3. R.S.Boyer, and J.S. Moore. A Fast String Searching Algorithm. *Comm ACM*, 20:762–772, 1977.

4. D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.
5. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
6. T. El-Gamal A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO'84,* Springer-Verlag (LNCS 196), pages 10–18, 1984.
7. U. Feige and A. Shamir. Zero Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO'89,* Springer-Verlag (LNCS 435), pages 526–544, 1989.
8. M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC 2005*, Springer-Verlag (LNCS 3378), pages 303–324, 2005.
9. M.J. Freedman, K. Nissim and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT 2004*, Springer-Verlag (LNCS 3027), pages 1–19, 2004.
10. O. Goldreich. Foundations of Cryptography: Volume 1 - Basic tools. Cambridge University Press, 2001.
11. O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.
12. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC,* pages 218–229, 1987.
13. S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90,* Springer-Verlag (LNCS 537), pages 77–93, 1990.
14. L. Kissner and D.X. Song. Privacy-Preserving Set Operations. In *CRYPTO 2005*, Springer-Verlag (LNCS 3621), pages 241–257, 2005.
15. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6:323–350, 1977.
16. Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology*, 16(3):143–184, 2003.
17. S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.
18. M. Naor and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. In 31*st STOC*, pages 245–254, 1999.
19. M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In 12*th SODA*, pages 448–457, 2001.
20. M. Naor and O. Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. In 38*th FOCS*, pages 231–262, 1997.
21. P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT '99*, Springer-Verlag (LNCS 1592), pages 223–238, 1999.
22. T. P. Pedersen. Non-Interactive and Information-Theoretical Secure Verifiable Secret Sharing. In *CRYPTO 1991*, Springer-Verlag (LNCS 576) pp. 129–140, 1991.
23. M. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.
24. A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.