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

Carmit Hazay† Yehuda Lindell†

September 14, 2008

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 are based on polynomials. 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. [6] following [15, 5, 22]) 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, 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 [28, 14, 13]. These feasibility results demonstrate the wide

*An extended abstract of this paper appeared in TCC 2008. This research was supported by an Eshkol scholarship and Infrastructures grant from the Israel Ministry of Science and Technology.

†Dept. of Computer Science, Bar-Ilan University, Israel. Email: {harelc,lindell}@cs.biu.ac.il.

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. Constructing such protocols is crucial if secure computation is ever to be used in practice.

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" [21].[1] 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 [14] we know 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 [24, 10] 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. However, it is highly suitable in the case that parties may suffer penalties if caught cheating (in addition, a potentially useful connection to the rational adversary model was pointed out by [17]). 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

[1] There are two main directions of research in privacy-preserving data mining. One area considers the problem of constructing secure protocols for distributed data mining, where security is in the sense of secure multiparty computation. The other area focuses on data privacy and uses techniques such as data perturbation [3], and is not related to our work here.

correctness are not achieved and they are sometimes needed. In order to see this, consider secure protocols for elections and auctions. Correctness is clearly crucial in elections to ensure that the candidate with the most votes is elected, and independence of inputs is needed in auctions in order to prevent an adversary from always winning by giving a bid that is only \$1 higher than the other bids. Despite the above, 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 without revealing anything else. 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 [23], 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 [18, 11], or together with random oracles when malicious adversaries are considered [11]. (We exclude the use of techniques that use general zero-knowledge proofs because these are not efficient.) We additionally note that the usage of polynomials requires $O(|X| \times |Y|)$ exponentiations without the ability to reduce it. Whereas in our case, the number of exponentiations depends *linearly* on the length of the inputs and the complexity of the oblivious PRF evaluation, which yields a construction with an improved efficiency.

Our protocols deal with the two-party set intersection problem. The multiparty set intersection problem is also of interest. We remark that our protocol does not seem to naturally extend to the multiparty setting.

Secure pattern matching. In addition to the above, we present an efficient secure protocol for solving the basic problem of *pattern matching* [4, 19]. 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 [25], 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. Our protocol does not extend to security in the presence of covert adversaries; see discussion at the end of Section 4.

Related work. The problem of secure set intersection was studied in [11] 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 [18] 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. A function $\mu(\cdot)$ is negligible in n (or just negligible) if for every polynomial $p(\cdot)$ there exists a value N such that for all $n > N$ it holds that $\mu(n) < \frac{1}{p(n)}$. Let $X = \{X(a,n)\}_{n\in N, a\in\{0,1\}^*}$ and $Y = \{Y(a,n)\}_{n\in N, a\in\{0,1\}^*}$ be distribution ensembles. Then, we say that X and Y are computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform distinguisher D there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$,

$$|\Pr[D(X(a,n)) = 1] - \Pr[D(Y(a,n)) = 1]| < \mu(n)$$

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 [12]. We also denote the ensemble of truly random functions by H_{Func} and the ensemble of truly random permutations by H_{Perm}.

2.1.1 Security in the Presence of Malicious Adversaries

In this section we briefly present the standard definition for secure multiparty computation and refer to [13, Chapter 7] for more details and motivating discussion.

Two-party computation. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-vector is a random variable $(f_1(x,y), f_2(x,y)$ ranging over pairs of strings where P_1 receives $f_1(x,y)$ and P_2 receives $f_2(x,y)$. We sometimes denote such a functionality by $(x,y) \mapsto (f_1(x,y), f_2(x,y))$. Thus, for example, the oblivious transfer functionality is denoted by $((x_0,x_1),\sigma) \mapsto (\lambda, x_\sigma)$, where (x_0,x_1) is the first party's input, σ is the second party's input, and λ denotes the empty string (meaning that the first party has no output).

Adversarial behavior. Loosely speaking, the aim of a secure multiparty protocol is to protect honest parties against dishonest behavior by other parties. In this section, we outline the definition for *malicious adversaries* who control some subset of the parties and may instruct them to arbitrarily deviate from the specified protocol. We also consider *static corruptions*, meaning that the set of corrupted parties is fixed at the onset.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation. One technical detail that arises when considering the setting of no honest majority is that it is impossible to achieve fairness or guaranteed output delivery [7]. That is, it is possible for the adversary to prevent the honest party from receiving outputs. Furthermore, it may even be possible for the adversary to receive output while the honest party does not. We consider malicious adversaries and static corruptions in this paper.

Execution in the ideal model. In an ideal execution, the parties send their inputs to the trusted party who computes the output. An honest party just sends the input that it received whereas a corrupted party can replace its input with any other value of the same length. Since we do not consider fairness, the trusted party first sends the output of the corrupted parties to the adversary, and the adversary then decides whether the honest parties receive their (correct) outputs or an abort symbol $\perp$. Let f be a two-party functionality where $f = (f_1, f_2)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subseteq [2]$ be the set of corrupted parties (either P_1 is corrupted or P_2 is corrupted or neither). Then, the ideal execution of f on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{IDEAL}_{f,\mathcal{A}(z),I}(x, y, n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. In the real model there is no trusted third party and the parties interact directly. The adversary $\mathcal{A}$ sends all messages in place of the the corrupted party, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of the specified protocol π.

Let f be as above and let π be a two-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the real execution of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{REAL}_{\pi,\mathcal{A}(z),I}(x, y, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol.

Definition 2.1 *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time 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 $I \subseteq [2]$,

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}$$

where $|x| = |y|$.

2.1.2 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. This notion of security has been considered in the past; see [24] for example. Formally, 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 2.2 *Let f be a 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 $\mathcal{A}$ controlling P_2 in the real model, there exists a non-uniform* PPT *adversary $\mathcal{S}$ for the ideal model, such that*

   $$\left\{\text{REAL}_{\pi,\mathcal{A}(z),2}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in N} \stackrel{\text{c}}{\equiv} \left\{\text{IDEAL}_{f,\mathcal{S}(z),2}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in N}$$

   where $|x| = |y|$.

2. *For every non-uniform* PPT *adversary $\mathcal{A}$ controlling P_1, and every polynomial $p(\cdot)$*

   $$\left\{\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),1}(x,y,n)\right\}_{x,y,y',z\in\{0,1\}^*,n\in N} \stackrel{\text{c}}{\equiv} \left\{\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),1}(x,y',n)\right\}_{x,y,y',z\in\{0,1\}^*,n\in N} \quad (1)$$

   where $|x| = |y| = |y'|$.

Note that the ensembles in Eq. (1) are indexed by two different inputs y and y' for P_2. The requirement is that $\mathcal{A}$ cannot distinguish between the case that P_2 used the first input y or the second input y'.

2.1.3 Security in the Presence of Covert Adversaries

Here we consider an adversary that 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 $\mathsf{corrupted}$ 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. The output distribution of an execution of this modified ideal model for a given ϵ and parameters as above is denoted $\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),i}(x,y,n)$. In more details, the ideal execution with ϵ proceeds as follows:

Inputs: P_1 receives input x and P_2 receives input y where $|x| = |y|$. The adversary receives an auxiliary-input z.

Send inputs to trusted party: The honest party sends its received input to the trusted party and the corrupted party, controlled by $\mathcal{A}$, may either send its received input or send some other input of the same length. Denote the pair of inputs sent to the trusted party by (w_1, w_2).

Abort options: If a corrupted party P_i sends $w_i = \mathsf{abort}_i$ to the trusted party as its input, then the trusted party sends abort_i to the honest party and halts. If a corrupted party P_i sends $w_i = \mathsf{corrupted}_i$ to the trusted party as its input, then the trusted party sends $\mathsf{corrupted}_i$ to the honest party and halts.

Attempted cheat option: If a corrupted party P_i sends $w_i = \mathsf{cheat}_i$ to the trusted party as its input, then the trusted party works as follows:

1. With probability ϵ, the trusted party sends $\mathsf{corrupted}_i$ to $\mathcal{A}$ and the honest party.
2. With probability $1-\epsilon$, the trusted party sends $\mathsf{undetected}$ to the adversary along with the honest party's input. Following this, the adversary sends the trusted party an output value z of its choice for the honest party. Then, the trusted party sends z to the honest party as its output.

The ideal execution then ends at this point.

If no w_i equals abort_i, $\mathsf{corrupted}_i$ or cheat_i, the ideal execution continues below.

Trusted party answers adversary: The trusted party computes $(f_1(w_1,w_2), f_2(w_1,w_2))$ and sends $\mathcal{A}$ the output of the corrupted party (i.e., for $I = \{i\}$, the trusted party sends $\mathcal{A}$ the value $f_i(w_1,w_2)$).

Trusted party answers the honest party: After receiving its output, the adversary sends either abort_i for some $i \in I$, or $\mathsf{continue}$ to the trusted party. If the trusted party receives $\mathsf{continue}$ then it sends $f_j(w_1,w_2)$ to the honest party P_j $(j \neq i)$. Otherwise, if it receives abort_i, it sends abort_i to the honest party.

Outputs: The honest party always outputs the message it obtained from the trusted party. The corrupted party outputs nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial input x_i to the corrupted party, the auxiliary input z, and the messages obtained from the trusted party.

The output of the honest parties and the adversary in an execution of the above ideal model is denoted by $\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),I}(x,y,n)$. We define:

Definition 2.3 *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 probabilistic polynomial-time 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 $I \subseteq [2]$,*

$$\left\{\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),I}(x,y,n)\right\}_{x,y,z,\in\{0,1\}^*,n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(x,y,n)\right\}_{x,y,z,\in\{0,1\}^*,n\in\mathbb{N}}$$

where $|x| = |y|$.

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 party P_1 may cause P_2 to receive an output that is not correctly computed without ever being caught.

2.1.4 Sequential Composition

Sequential composition theorems for secure computation are important for two reasons. First, they constitute a security goal within themselves. Second, they are useful tools that help in writing proofs of security. The basic idea behind these composition theorems is that it is possible to design a protocol that uses an ideal functionality as a subroutine, and then analyze the security of the protocol when a trusted party computes this functionality. For example, assume that a protocol is constructed that uses the secure computation of some functionality as a subroutine. Then, first we construct a protocol for the functionality in question and prove its security. Next, we prove the security of the larger protocol that uses the functionality as a subroutine in a model where the parties have access to a trusted party computing the functionality. The composition theorem then states that when the "ideal calls" to the trusted party for the functionality are replaced by real executions of a secure protocol computing this functionality, the protocol remains secure.

The hybrid model. The aforementioned composition theorems are formalized by considering a *hybrid model* where parties both interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, the parties run a protocol π that contains "ideal calls" to a trusted party computing some functionalities $f_1, \ldots, f_m$. These ideal calls are just instructions to send an input to the trusted party. Upon receiving the output back from the trusted party, the protocol π continues. We stress that honest parties do not send messages in π between the

time that they send input to the trusted party and the time that they receive back output (this is because we consider *sequential* composition here). Of course, the trusted party may be used a number of times throughout the π-execution. However, each time is independent (i.e., the trusted party does not maintain any state between these calls). We call the regular messages of π that are sent amongst the parties standard messages and the messages that are sent between parties and the trusted party ideal messages.

Let $f_1, \ldots, f_m$ be probabilistic polynomial-time functionalities and let π be a two-party protocol that uses ideal calls to a trusted party computing $f_1, \ldots, f_m$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the $f_1, \ldots, f_m$-hybrid execution of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}(z),I}(x, y, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing $f_1, \ldots, f_m$.

Sequential modular composition. Let $f_1, \ldots, f_m$ and π be as above, and let $\rho_1, \ldots, \rho_m$ be protocols. Consider the real protocol $\pi^{\rho_1,\ldots,\rho_m}$ that is defined as follows. All standard messages of π are unchanged. When a party P_i is instructed to send an ideal message α_i to the trusted party to compute functionality f_j, it begins a real execution of ρ_j with input α_i instead. When this execution of ρ_j concludes with output β_i, party P_i continues with π as if β_i was the output received by the trusted party (i.e. as if it were running in the $f_1, \ldots, f_m$-hybrid model). Then, the composition theorem of [6] states that if ρ_j securely computes f_j for every $j \in \{1, \ldots, m\}$, then the output distribution of a protocol π in a hybrid execution with $f_1, \ldots, f_m$ is computationally indistinguishable from the output distribution of the real protocol $\pi^{\rho_1,\ldots,\rho_m}$. This holds for security in the presence of malicious adversaries [6], one-sided simulation when considering the corruption case that has a simulator (an easy corollary from [6]), and security in the presence of covert adversaries (see [2]). We refer the reader to [2] and [6] for formal statements of the composition theorem.

2.2 Tools

In this section, we consider two basic tools used in our constructions; oblivious transfer and oblivious pseudorandom function, for which we present the construction of Naor-Reingold [25] that uses oblivious transfer.

2.2.1 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. Our protocols can use any oblivious transfer subprotocol that achieves the appropriate level of security (one-sided simulatability, covert or full security in the presence of malicious adversaries). For example, we can use the construction of [16] that is fully simulatable and achieves Definition 2.1. This construction is based on the protocols of [1, 24] and only requires 26 exponentiations per execution.

Batched oblivious transfers. We remark that our protocols actually need to run multiple oblivious transfers in parallel. For the sake of this, we define the *batched oblivious transfer functionality* with m executions, denoted $\mathcal{F}^m_{\text{OT}}$ as follows:

$$((x^0_1, x^1_1), \ldots, (x^0_m, x^1_m), (i_1, \ldots, i_m)) \rightarrow (\lambda, (x^{i_1}_1, \ldots, x^{i_m}_m))$$

A fully simulatable protocol that securely computes $\mathcal{F}_{\text{OT}}^m$ in the presence of malicious adversaries can be found in [16]. The protocol runs in a constant number of rounds, requires $14m + 14$ exponentiations for all m executions, and is of comparative complexity to [1] (with the advantage of full simulatability). When considering covert adversaries, we can use the batched oblivious transfer construction from [2] that requires on average four exponentiations per transfer.

2.2.2 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

$$(k, x) \mapsto (\lambda, F_{\text{PRF}}(k, x)) \tag{2}$$

where $k \leftarrow I_{\text{PRF}}(1^n)$ and $x \in \{0,1\}^n$.[2] We will use the Naor-Reingold [25] pseudorandom function ensemble F_{PRF} (with some minor modifications). For every n, 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 [25] 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 [10] and involves the parties running an oblivious transfer execution for every bit of the input x. For the sake of completeness, we provide a formal description and analysis, explicitly dealing with security in the presence of malicious adversaries (with simulation), security in the presence of covert adversaries, and one-sided simulation. As we will see, the only difference between the different levels of security is the security of the oblivious transfer protocol used. The protocol follows.

Protocol $\boldsymbol{\pi_{\text{PRF}}}$

- **Inputs:** The input of P_1 is $k = (p, q, g^{a_0}, a_1, \ldots, a_n)$ and the input of P_2 is a value x of length n.
- **Auxiliary inputs:** Both parties have the security parameter 1^n and are given the primes p and q.
- **The protocol:**
  1. P_1 chooses n random values $r_1, \ldots, r_n \in_R \mathsf{Z}_q^*$.
  2. The parties engage in a 1-out-2 multi-oblivious transfer protocol π_{OT}^m (with $m = n$ executions). In the ith iteration, P_1 inputs $y_0^i = r_i$ and $y_1^i = r_i \cdot a_i$ (with multiplication in Z_q^*), and P_2 enters the bit $\sigma_i = x_i$ where $x = x_1, \ldots, x_n$. If the output of any of the oblivious transfers is $\perp$, then both parties output $\perp$ and halt. Otherwise:

[2] 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.

3. P_2's output from the n executions is a series of values $y^1_{x_1}, \ldots, y^n_{x_n}$. If any value $y^i_{x_i}$ is not in Z^*_q, then P_2 redefines it to equal 1.
4. P_1 computes $\tilde{g} = g^{a_0 \cdot \prod_{i=1}^n \frac{1}{r_i}}$ and sends it to P_2.
5. P_2 aborts if the order of $\tilde{g}$ is different than q. Otherwise, P_2 computes $y = \tilde{g}^{\prod_{i=1}^n y^i_{x_i}}$ and outputs y.

Before proceeding, note that if P_1 and P_2 follow the instructions of the protocol, then the output of P_2 is:

$$y = \tilde{g}^{\prod_{i=1}^n y^i_{x_i}} = g^{a_0 \cdot \prod_{i=1}^n \frac{y^i_{x_i}}{r_i}} = g^{a_0 \cdot \prod_{i=1}^n a_i^{x_i}} = F_{\text{PRF}}(k, x)$$

where the second last equality is due to the fact that for $x_i = 0$ it holds that $a_i^{x_i} = 1$ and $y^i_{x_i}/r_i = y^i_0/r_i = 1$, and for $x_i = 1$ it holds that $a_i^{x_i} = a_i$ and $y^i_{x_i}/r_i = y^i_1/r_i = a_i$. We now prove security:

Proposition 2.4 *Assume that π^m_{OT} securely computes the multi-oblivious transfer functionality in the presence of malicious adversaries and that the DDH assumption holds in the subgroup generated by g. Then π_{PRF} securely computes $\mathcal{F}_{\text{PRF}}$ in the presence of malicious adversaries.*

Proof: We separately analyze the case that P_1 is corrupted and the case that P_2 is corrupted. We prove the proposition in the hybrid model where a trusted party is used to compute the oblivious transfers; see Section 2.1.4.

P_1 **is corrupted.** Let $\mathcal{A}$ be an adversary controlling P_1. We construct a simulator $\mathcal{S}$ as follows. $\mathcal{S}$ receives $\mathcal{A}$'s inputs for all the n iterations of the oblivious transfers (recall our analysis is in the hybrid model). Let y_{i_0} and y_{i_1} denote the inputs that $\mathcal{A}$ handed $\mathcal{S}$ in the ith iteration. In addition, $\mathcal{S}$ receives from $\mathcal{A}$ the message $\tilde{g}$. In case $\mathcal{A}$ does not send a valid message (where $\mathcal{S}$ conducts the same check as the honest P_2 does), $\mathcal{S}$ simulates P_2 aborting and sends $\perp$ to the trusted party. Otherwise, $\mathcal{S}$ checks the validity of all the y^i_0 and y^i_1 values and modifies them to 1 if necessary (as would an honest P_2). $\mathcal{S}$ defines $g_0 = \tilde{g}$. Then, for every $i = 1, \ldots, n$, $\mathcal{S}$ defines:

$$a_i = \frac{y^i_1}{y^i_0} \quad \text{and} \quad g_i = (g_{i-1})^{y^i_0}$$

$\mathcal{S}$ defines the key used by $\mathcal{A}$ to be $k = (p, q, g_n, a_1, \ldots, a_n)$ and sends it to $\mathcal{F}_{\text{PRF}}$. This completes the description of $\mathcal{S}$. It is immediate that the view of $\mathcal{A}$ is identical in a real and simulated execution because it receives no messages in a hybrid execution where the oblivious transfers are run by a trusted party. It thus remains to show that the output received by P_2 in a real execution is the same as in the ideal model. In order to see this, first note that $\mathcal{S}$ and P_2 replace any invalid y^i_0 or y^i_1 values in the same way. Next, note that for any $x \in \{0,1\}^n$:

$$F_{\text{PRF}}(k, x) = g_n^{\prod_{i=1}^n a_i^{x_i}} = \tilde{g}^{\prod_{i=1}^n y^i_0 \cdot a_i^{x_i}} = \tilde{g}^{\prod_{i=1}^n y^i_0/(y^i_0)^{x_i} \cdot (y^i_1)^{x_i}}$$

where the first equality is by the definition of the key by $\mathcal{S}$, the second equality is by the fact that $g_n = \tilde{g}^{\prod_{i=1}^n y^i_0}$, and the third equality is by the fact that $a_i = y^i_1/y^i_0$. Notice now that if $x_i = 0$ we have that $y^i_0/(y^i_0)^{x_i} \cdot (y^i_1)^{x_i} = y^i_0$, whereas if $x_i = 1$ we have that $y^i_0/(y^i_0)^{x_i} \cdot (y^i_1)^{x_i} = y^i_1$. Thus, $\prod_{i=1}^n y^i_0/(y^i_0)^{x_i} \cdot (y^i_1)^{x_i} = \prod_{i=1}^n y^i_{x_i}$, exactly as computed by P_2 in a real execution. That is, the computation of $F_{\text{PRF}}(k, x)$ as carried out by the trusted party using the key supplied by $\mathcal{S}$ is the same as that obtained by P_2 in a real execution. This completes the case that P_1 is corrupted.

P_2 **is corrupted.** In this case, $\mathcal{S}$ learns the full input x of $\mathcal{A}$ controlling P_2 (through the oblivious transfer inputs). In each oblivious transfer, $\mathcal{S}$ hands $\mathcal{A}$ a random value $r_i \in_R \mathsf{Z}_q^*$. After all of the oblivious transfers have concluded, $\mathcal{S}$ sends x to $\mathcal{F}_{\text{PRF}}$ and receives back a value $y = F_{\text{PRF}}(k, x)$. $\mathcal{S}$ then sets $\tilde{g} = y^{\prod_{i=1}^{n} \frac{1}{r_i}}$ and sends it to $\mathcal{A}$. This completes the simulation.

We claim that the view of $\mathcal{A}$ in an execution of π_{PRF} with P_1 (using a trusted party for the oblivious transfers) is identical to its view in an ideal execution with $\mathcal{S}$. This is true because all of the r_i values are distributed identically to the messages sent in a real execution (note that r_i and $r_i \cdot a_i$ have the same distribution). In particular, giving $\tilde{g}$ and all the r_i values is it easy to define a valid random PRF key. Furthermore, in a real execution, it holds that $\tilde{g}^{\prod_{i=1}^{n} y_{x_i}^i} = g^{a_0 \prod_{i=1}^{n} a_i^{x_i}} = y$, where the x_i values are those used by P_2 in the oblivious transfers. Likewise, in the simulation it holds that

$$\tilde{g}^{\prod_{i=1}^{n} y_{x_i}^i} = \left(y^{\prod_{i=1}^{n} \frac{1}{r_i}}\right)^{\prod_{i=1}^{n} r_i} = y = g^{a_0 \prod_{i=1}^{n} a_i^{x_i}}$$

where first equality is due to the fact that the simulator sets each y^i value received by $\mathcal{A}$ to r_i, and the last equality is by the fact that y is computed correctly by the trusted party. Thus, the joint distribution over the values received by $\mathcal{A}$ and $\tilde{g}$ in the hybrid and ideal executions are exactly the same. Formally,

$$\{\text{IDEAL}_{\mathcal{F}_{\text{PRF}}, \mathcal{S}(z), 2}(X, Y, n)\} \equiv \{\text{HYBRID}^{\text{OT}}_{\pi_{\text{PRF}}, \mathcal{A}(z), 2}(X, Y, n)\}$$

and the proof is concluded. ■

Security with one-sided simulation and in the presence of covert adversaries. Almost identical proofs yield the following proposition:

Proposition 2.5 *Assume that the DDH assumption holds in the subgroup generated by g, and assume that π_{OT}^m securely computes the multi-oblivious transfer functionality in the presence of covert adversaries with deterrent ϵ (resp., is secure under one-sided simulation). Then π_{PRF} securely computes $\mathcal{F}_{\text{PRF}}$ in the presence of covert adversaries with deterrent ϵ (resp., is secure under one-sided simulation).*

Multi-execution protocol. We remark that if the oblivious transfers that are used can be run simultaneously (or batched), as with the simultaneous oblivious transfer of [2, 16], then we can run many executions of π_{PRF} simultaneously. This is of great importance for efficiency. Using the oblivious transfer of [2] we 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, and using the oblivious transfer of [16], we have $14\ell + 14$ exponentiations.

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. (2) 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 3.1 *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: 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, where the ability to break the security of a single execution is reduced to the ability to break the security of multiple executions.

More formally, assume by contradiction that there exist a probabilistic polynomial-time adversary $\mathcal{A}$, and probabilistic polynomial-time distinguisher D and infinitely many input sets X, $Y = \{y_1, \ldots, y_{m_2}\}$ and $Y' = \{y'_1, \ldots, y'_{m_2}\}$ (subjected to the length constraints), such that D distinguishes between the views of $\mathcal{A}$ when running π_{INT} with P_2 who has input Y, and when running π_{INT} with P_2 who has input Y'. Then we show that there exists an adversary $\mathcal{A}'$ and infinitely many pairs of inputs y, y' such that the view of $\mathcal{A}$ when running π_{PRF} with P_2 who has input y can be distinguished from the view of $\mathcal{A}$ when running π_{PRF} with P_2 who has input y'. Let H_i denote

the view of $\mathcal{A}$ such that the first i elements in the input of P_2 are $y_1, \ldots, y_i$ and the last $m_2 - i$ elements are $y'_{i+1}, \ldots, y'_{m_2}$. Using a standard hybrid argument, there exists an index i such that the distributions H_i and H_{i+1} are distinguishable. Then we construct an adversary $\mathcal{A}_{\text{PRF}}$ that controls P_1 and runs against an external P_2, such that P_2's input is either y_{i+1} or y'_{i+1}, and a distinguisher D_{PRF} that breaks the security of a single execution of π_{PRF}. Fix Y, Y' and X. Then on input 1^n and auxiliary input Y, Y', z, X and i, $\mathcal{A}_{\text{PRF}}$ does the following: it invokes $\mathcal{A}$ on input $1^n, X$ and auxiliary input z and plays the role of the honest P_2 for all but execution $i+1$ of the oblivious PRF evaluation. That is, $\mathcal{A}_{\text{PRF}}$ internally runs P_2 for all but the $(i+1)$th execution: for $j \leq i$ it uses y_j and for $j > i+1$ it uses y'_j, whereas for the $(i+1)$th execution, $\mathcal{A}_{\text{PRF}}$ forwards all the messages between $\mathcal{A}$ and the external P_2, and outputs whatever $\mathcal{A}$ does (note that the PRF executions are independent from P_2's viewpoint). Finally, D_{PRF} invokes D on the output of $\mathcal{A}_{\text{PRF}}$ and returns its output. Clearly, if the external P_2 uses input y_{i+1}, the view of $\mathcal{A}$ is distributed as in H_{i+1}, whereas if it uses y'_i then this yields a distribution identical to H_i. Thus the non-negligible distinguishing gap of D can be reduced to breaking the privacy property of π_{PRF}.

P_2 is corrupted. 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 $Y = \{y_1, \ldots, y_{m_2}\}$ to the pseudorandom function evaluations. $\mathcal{S}$ chooses a unique random value ζ_i for each distinct y_i, hands it to $\mathcal{A}$ as its output in the ith evaluation and records the pair (y_i, ζ_i). $\mathcal{S}$ then sends Y 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 R 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 $\mathcal{A}$.[3] Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. Note that the difference between the hybrid and the simulated executions is due to the fact that $\mathcal{S}$ provides random values rather than pseudorandom ones (this is equivalent to saying that $\mathcal{S}$ computes $\zeta_i = H_{\text{Func}}(y_i)$ for every i). In addition, the set R that represents $X - (X \cap Y)$ is random as well. We complete the proof through the following series of games;

Game H_1: We begin by modifying $\mathcal{S}$ so that it uses an oracle $\mathcal{O}_{H_{\text{Func}}}$ instead of computing the function H_{Func}. That is, $\mathcal{S}$ sends its oracle the sets Y and R. By the definition of H_{Func}, this is exactly the same distribution as generated by $\mathcal{S}$ above.

Game H_2: We now modify $\mathcal{S}$ so that it uses the real input X of P_1. The resulting distribution is identical because the oracle computes a truly random function and all inputs are distinct in both cases.

Game H_3: Next, we replace the oracle $\mathcal{O}_{H_{\text{Func}}}$ with an oracle $\mathcal{O}_{F_{\text{PRF}}}$ computing F_{PRF}. Clearly, the resulting distributions in both games are computationally indistinguishable. This can be proven via a reduction to the pseudorandomness of the function F_{PRF}.

Informally, let D_{PRF} denote a distinguisher who attempts to distinguish F_{PRF} from H_{Func}. Then D_{PRF}, playing the roles of P_1 as $\mathcal{S}$ above, invokes its oracle on the sets Y and R. Now, any distinguisher for the distributions of games H_2 and H_3 can be utilized by D_{PRF} to distinguish between F_{PRF} and H_{Func}.

[3] 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.

Game H_4: Finally, $\mathcal{S}$ computes the pseudorandom function instead of using an oracle. Again, this makes no difference whatsoever for the output distribution.

Noting that the last game is exactly the distribution generated in a hybrid execution, we have that the hybrid and ideal executions are computationally indistinguishable, completing the proof. ■

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 (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). 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 [18] (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 protocol for proving this is known.)

3.2 Secure Set Intersection in the Presence of Covert Adversaries

In this section we present a protocol for securely computing the two-party set-intersection functionality 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_1 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 the dummy evaluations (this means that if P_1 attempts to cheat by constructing V_0 or V_1 incorrectly it will be caught with probability 1/2). 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. We remark that security in the case that P_2 is corrupted is of the same level as the subprotocols used for oblivious transfer, coin tossing and pseudorandom function evaluation (i.e., full security if the subprotocols are fully secure in the presence of malicious adversaries and covert in the case that they are secure in the presence of covert adversaries).

We stress that only P_2 receives output in our protocol. In order to have P_1 also receive output, we cannot have P_2 just send the values in the intersection. This is due to the fact that P_2 can omit values from the intersection that it sends back to P_1. (Note that it is not so difficult to have P_2 prove that every value returned to P_1 is in the intersection. However, it seems much harder to prevent P_2 from omitting values.)

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:

- An efficiently computable and invertible pseudorandom permutation with sampling algorithm I_{PRP}; see [12, Chapter 2]. 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 function 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, denoted by π_{CT}, that is secure in the presence of covert adversaries with deterrent $\epsilon = 1/2$. 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. Such a protocol can be constructed by an instantiation of the generic coin-tossing protocol that appears in [20], with commitments based on El-Gamal encryption [8]. The instantiation of El-Gamal enables highly efficient zero-knowledge proofs of knowledge with soundness 1/2 for the discrete logarithm and Diffie-Hellman tuple languages (which suffice for achieving security in the covert model with deterrent $\epsilon = 1/2$).

  In particular, in [20] the committer first sends a commitment $c = \mathsf{com}(s_1; r_1)$ for $s_1 \in_R \{0,1\}^n$, and then the parties engage in a zero-knowledge argument of knowledge in which P_1 proves it knows s_1. Next, P_2 chooses $s_2 \in_R \{0,1\}^n$ and sends it to P_1. Then P_1 outputs s_1, s_2

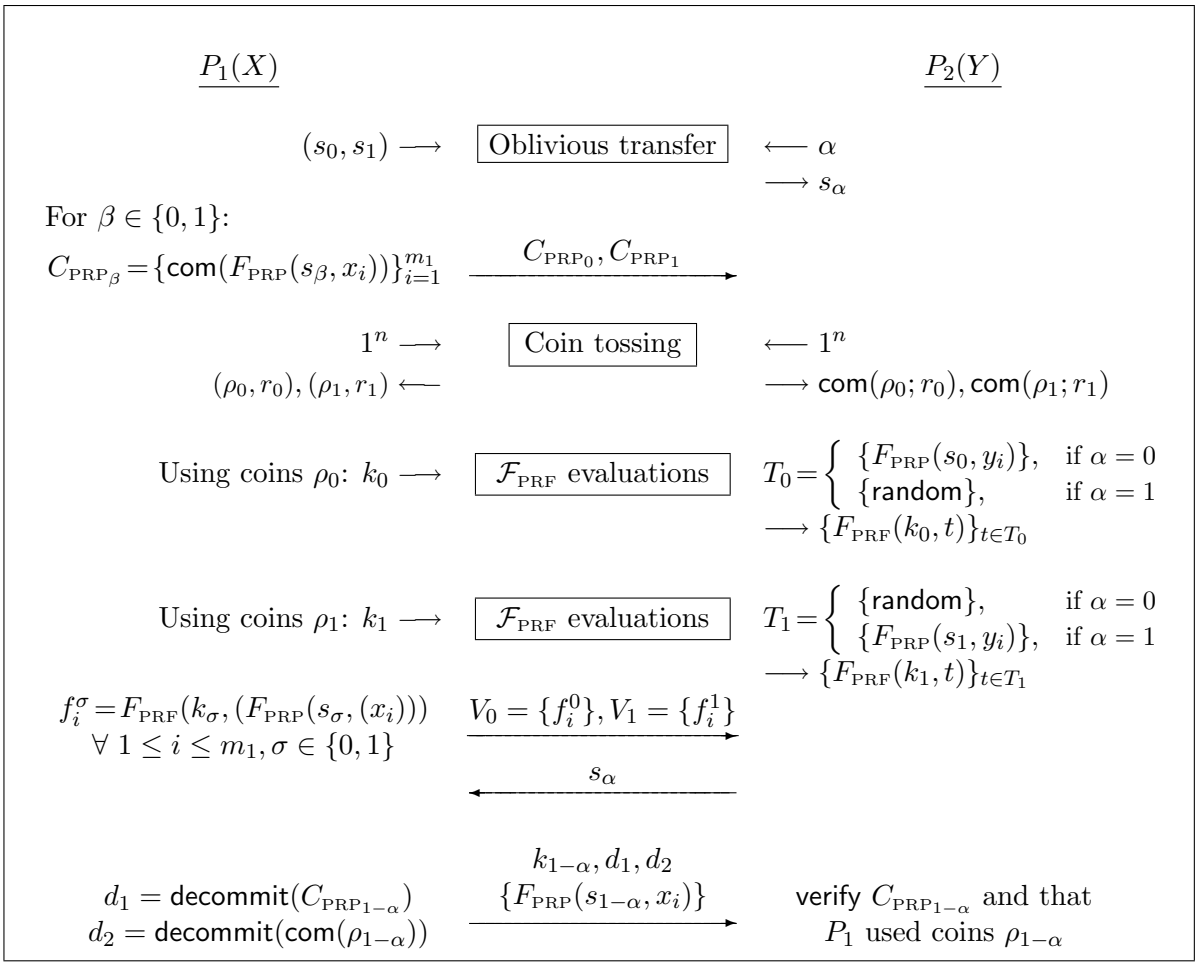

Figure 1: A high-level diagram of our protocol.

and r_1, and P_2 outputs c and s_2. For the particular instantiation of El-Gamal, recall that its public-key is $\langle p, q, g, g^\alpha \rangle$ and the corresponding private-key is $\langle p, q, g, \alpha \rangle$. Thus a proof of knowledge of the committed value s_1 can be easily achieved if the committer proves the knowledge of α, since then the committed value can be decrypted.

We stress that it suffices to generate a short random value (i.e. one random group element), and then to use a pseudorandom generator to extend it.

- A protocol π_{PRF} for computing $\mathcal{F}_{\mathrm{PRF}}$ as defined in Eq. (2), 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 $s_0, s_1 \leftarrow \mathcal{I}_{\text{PRP}}(1^{p(n)})$ 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 $C_{\text{PRP}_0} = \{\mathsf{com}(F_{\text{PRP}}(s_0, x))\}_{x \in X}$, $C_{\text{PRP}_1} = \{\mathsf{com}(F_{\text{PRP}}(s_1, x))\}_{x \in X}$ and sends C_{PRP_0} and C_{PRP_1} to P_2.
3. The parties run two executions of the coin-tossing protocol π_{CT} computing $(1^{q(n)}, 1^{q(n)}) \rightarrow ((\rho, r), \mathsf{com}(\rho; r))$ that is secure for covert adversaries with $\epsilon = 1/2$. The parties input $1^{q(n)}$, where $q(n)$ is the number of random bits needed to both choose a key $k \leftarrow I_{\text{PRF}}(1^{p(n)})$ 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 $k_0 \leftarrow \mathcal{I}_{\text{PRF}}(1^{p(n)})$ in each execution, and P_2 enters the elements of the set $T_0 = \{F_{\text{PRP}}(s_0, y)\}_{y \in Y}$ (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 $k_1 \leftarrow \mathcal{I}_{\text{PRF}}(1^{p(n)})$ each time, and P_2 enters m_2 random values of size $p(n)$ (if $\alpha = 0$), and the elements of the set $T_1 = \{F_{\text{PRP}}(s_1, y)\}_{y \in Y}$ (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 decommitted values in $C_{\text{PRP}_{1-\alpha}}$ and $\rho_{1-\alpha}$ the decommitted values in 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 3.2 *Assume that $\pi_{\text{OT}}, \pi_{\text{CT}}, \pi_{\text{PRF}}$ are secure in the presence of covert adversaries with deterrent $\epsilon = \frac{1}{2}$, 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 both parties are honest, the case that P_1 is corrupted and the case that P_2 is corrupted. A joint simulator can be constructed on the basis of these cases. 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 functions by $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{CT}}$. Since the last step of the protocol involves P_2 checking the actual messages sent by P_1 in the PRF evaluations, we cannot replace the PRF evaluations with ideal executions invoking a trusted party. We do, however, use the simulator $\mathcal{S}_{\text{PRF}}$ that is assumed to exist for this protocol (we assume that the PRF evaluation protocols are all run simultaneously and so $\mathcal{S}_{\text{PRF}}$ can simulate them all together).

No corruptions. Assume both parties are honest (i.e, $\mathcal{I} = \phi$). Note that in the ideal calls to $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{CT}}$, the adversary sees nothing when no party is corrupted. The PRF evaluations can be simulated by running $\mathcal{S}_{\text{PRF}}$ for the case of no corrupted parties (and so we ignore these from here on). Thus, prior to the last step of checks, the only messages that remain to be simulated are the commitment sets C_{PRP_0} and C_{PRP_1} and the sets of pseudorandom values V_0 and V_1, sent by P_1 to P_2. Then, in the last step, P_2 sends a random string s_α to P_1, and P_1 replies with decommitments to $C_{\text{PRP}_{1-\alpha}}$ and the decommitment of $c_{\rho_{1-\alpha}}$. However, $C_{\text{PRP}_{1-\alpha}}$ is a set of values obtained by applying a pseudorandom permutation keyed by an unknown $s_{1-\alpha}$. A straightforward reduction to the pseudorandom function and permutation shows that a view generated by $\mathcal{S}$ by just sending random values (from the appropriate range) instead of pseudorandom ones, yields a view that is indistinguishable from a real one. This is standard, and so details are omitted. (We remark that the simulator $\mathcal{S}$ is given the sizes of the sets as auxiliary input and so can carry out the simulation.)

The above relates to $\mathcal{A}$'s view of the transcript as generated by $\mathcal{S}$ versus a real execution. However, we also need to show that the outputs of the honest parties are indistinguishable in a real and ideal execution; that is, we need to prove *correctness*. Now, for every $\zeta \in X \cap Y$, both honest parties obtain the same value $F_{\text{PRF}}(k_\alpha, (F_{\text{PRP}}(s_\alpha, \zeta))$ and so P_2 records ζ as part of its output. Thus, the output of P_2 in a real execution includes at least every value in the intersection. It remains to show that it does not include any additional values. This can occur if there exists an $x \in X$ and $y \in Y$ for which $x \neq y$ but $F_{\text{PRF}}(k_\alpha, (F_{\text{PRP}}(s_\alpha, x)) = F_{\text{PRF}}(k_\alpha, (F_{\text{PRP}}(s_\alpha, y))$. However, since s_α and k_α are uniformly chosen, the probability that any two fixed values x and y collide in this way is negligible (this follows from the fact that such a collision occurs with a random function with negligible probability). Since there are a polynomial number of pairs of values overall, it follows that P_2 outputs $y \notin X \cap Y$ with at most negligible probability.

Party P_1 is corrupted. Let $\mathcal{A}$ be an adversary controlling party P_1, we informally describe a simulator $\mathcal{S}$ as follows. At the beginning of the simulation, $\mathcal{S}$ learns the adversary's inputs s_0 and s_1 to the oblivious transfer execution, which are used as keys for the PRP. The knowledge of this pair enables the simulator to learn both (W_0, k_0) and (W_1, k_1) by rewinding $\mathcal{A}$ in Step 6 of the protocol. This in turn enables the simulator to extract $\mathcal{A}$'s inputs by computing $\{F^{-1}_{\text{PRP}}(s_\alpha, w)\}_{w \in W_\alpha}$; note that F_{PRP} is efficiently invertible. We note that if the adversary aborts without responding correctly to any of the queries in Step 6, then the honest P_2 aborts in the real execution and so the simulator just sends $\perp$ to the trusted party. In contrast, if the adversary responds to only one query correctly, then the simulator aborts with probability half exactly as in a real execution. Formally,

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 computing $\mathcal{F}_{\text{OT}}$:

(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 its trusted party computing $\mathcal{F}_\cap$. 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 computing $\mathcal{F}_\cap$ (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.[4]

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 computing $\mathcal{F}_\cap$, 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 (note that the checks regarding the behavior of $\mathcal{A}$ in the PRF evaluations can be run even though $\mathcal{S}$ used $\mathcal{S}_{\text{PRF}}$ because the check just involves verifying the responses of P_1 to the messages that it received in these executions.) We have two cases:

(a) *Case 1 – all of the checks carried out 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 computing $\mathcal{F}_\cap$ the set $\{F^{-1}_{\text{PRP}}(s_\alpha, w)\}_{w \in W_\alpha}$ (recall that we assume that F_{PRP} can be efficiently invertible).

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

(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 computing $\mathcal{F}_\cap$. 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. (We note that in a real execution, P_2 uses its input in the $\mathcal{F}_{\text{PRF}}$ evaluations. However, $\mathcal{S}$ received all of the keys used by $\mathcal{A}$ in these executions from what $\mathcal{S}_{\text{PRF}}$ intended to send to the trusted party computing $\mathcal{F}_{\text{PRF}}$ in its simulation. Thus, $\mathcal{S}$ can compute the outputs that P_2 would have received even if $\mathcal{A}$ used different keys in the different executions.) $\mathcal{S}$ then sends the trusted party computing $\mathcal{F}_\cap$ 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

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

where the ensembles are indexed by $X \subseteq \{0,1\}^{p(n)}$ of size m_1, $Y \subseteq \{0,1\}^{p(n)}$ of size m_2, $z \in \{0,1\}^*$ and $n \in \mathbb{N}$. 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 because $\mathcal{A}$ receives no output (playing the role of P_1) and so the security of π_{PRF} implies that $\mathcal{A}$ cannot distinguish between a real execution with P_2 using real inputs and a simulation with $\mathcal{S}_{\text{PRF}}$ with no input at all. 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 that occur with almost the same probability in both executions, due to the indistinguishable views. 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. Now, since $\mathcal{S}$ forwards $\mathsf{corrupted}_1$ and abort_1 to the trusted party computing $\mathcal{F}_\cap$, P_2 outputs these messages with the exact same probability in both executions. As for cheat_1, whenever $\mathcal{A}$ sends this message during the above executions, if $\mathcal{S}$ receives back $\mathsf{corrupted}_1$ then the simulation immediately halts. Furthermore, if it receives back $\mathsf{undetected}$ then the simulator also receives the real input Y of P_2 from the trusted party computing $\mathcal{F}_\cap$ and is able to complete the execution emulating the honest P_2 (note that the messages until this point of the execution are independent of P_2's input and so $\mathcal{S}$ can conclude the execution consistently with the input set Y). At the end of this execution (using P_2's real input Y), $\mathcal{S}$ obtains the output that P_2 would obtain in a real execution with $\mathcal{A}$ and sends it to the trusted party computing $\mathcal{F}_\cap$ to be P_2's output in the ideal model. Thus, the output distributions are identical. That is,

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

In the second case, $\mathcal{A}$ provides valid inputs for $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{CT}}$, yet there exists an $\alpha \in \{0,1\}$ 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). This holds because all of the parameters for these computations are already set without $\mathcal{A}$ having the ability to change them (i.e., $\mathcal{A}$ must use

the coins $\rho_{1-\alpha}$ to choose the key $k_{1-\alpha}$ and run all of the π_{PRF} evaluations that use this key). Then, after sending s_α to P_1, party P_2 is given $\rho_{1-\alpha}$ and is able to recompute the entire transcript of the PRF evaluations in order to compare it against the actual messages it received during its interaction with $\mathcal{A}$. Thus, in both the hybrid and ideal executions, P_2 outputs $\mathsf{corrupted}_1$ with probability at least $\frac{1}{2}$. In particular, if $\mathcal{A}$ provides two invalid responses it is always caught, and P_2 always outputs $\mathsf{corrupted}_1$ (recall that in the ideal execution, $\mathcal{S}$ sends $\mathsf{corrupted}_1$ to the trusted party computing $\mathcal{F}_\cap$). Whereas, if $\mathcal{A}$ provides exactly one valid response, then P_2 outputs $\mathsf{corrupted}_1$ with probability $\frac{1}{2}$. 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 and s_0, s_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 π_{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 $\neg\mathsf{bad}$. In this case the simulator does not have access to Y and needs to fully extract $\mathcal{A}$'s input. 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 3.3 *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 $\zeta \in X_\alpha \cap Y$ if and only if $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, \zeta)) \in V_\alpha \cap U_\alpha$, except with negligible probability.*

Proof: If $\zeta \in X_\alpha \cap Y$, then $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, \zeta)) \in V_\alpha \cap U_\alpha$. This is due to the fact that $\mathcal{A}$ uses the same key k_α for the PRF evaluation that defines U_α and for computing V_α. In particular, we consider here the case where $\mathcal{A}$ provides valid responses for both $\alpha = 0$ and $\alpha = 1$, combined with the fact that it can be verified that $\mathcal{A}$ indeed used k_α for computing U_α. This implies that if $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, \zeta)) \notin V_\alpha \cap U_\alpha$, then it must be that there exists an $\alpha \in \{0,1\}$ for which $\mathcal{A}$ cannot provide a valid response in Step 6, and this is a contradiction.

As for the other direction, assume that $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, \zeta)) \in V_\alpha \cap U_\alpha$. 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). More formally, assume that there exists an infinite series of input sets Y, and a PPT adversary $\mathcal{A}$ such that the probability that there exists a value $x \in X$ where $F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, x)) \in V_\alpha \cap U_\alpha$ yet $x \notin X \cap Y$ is non-negligible. Then we construct a distinguisher D_{PRF} that distinguishes F_{PRF} from a truly random function H_{Func} given an oracle $\mathcal{O}$ that computes one of these functions. Fix Y and let (Y, z) denote D_{PRF}'s auxiliary input. Then $D_{\text{PRF}}(1^n)$ invokes $\mathcal{A}$ on $(1^n, z)$ and plays the honest $P_2(1^n, Y)$ until the point where $\mathcal{A}$ opens its input commitments C_{PRP_α} into the values $F = (f_1, \ldots, f_{M_1})$ at step 6c (note $\mathcal{A}$ opens these commitments always since we consider the case where $\mathcal{A}$ is not caught cheating). Then D_{PRF} sends its oracle the sets $\{\{F_{\text{PRP}}(s_\alpha, y)\}_{y \in Y}$ and F and outputs 1 if and only if there exists a collision between the sets (i.e., a pair of distinct values $u \in \{\{F_{\text{PRP}}(s_\alpha, y)\}_{y \in Y}$ and $v \in F$ for which $\mathcal{O}(u) = \mathcal{O}(v)$). Note that

if $\mathcal{O}$ is a truly random function, then the probability that D_{PRF} outputs 1 is negligible. On the other hand, if $\mathcal{O}$ computes F_{PRF}, then by our assumption the probability that D_{PRF} outputs 1 is non-negligible. This holds because the question of whether or not there is a collision depends only on the committed set provided by $\mathcal{A}$ (that defines F), the value s_α, and the honest party's input Y. All of this is fixed before k_α is chosen (via the *coin tossing*) and so the probability of a collision depends only on the choice of the key k_α. Since this key is uniformly chosen, a collision occurs in the oracle with the same probability that it occurs in the hybrid model (with a trusted party computing the coin tossing functionality).[5] Thus, we have that D_{PRF} distinguishes between F_{PRF} and a random function with non-negligible probability, in contradiction to the security of F_{PRF}. ■

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:

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

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. Intuitively, the simulator $\mathcal{S}$ works as follows. First, it learns $\mathcal{A}$'s input α to the oblivious transfer execution which enables $\mathcal{S}$ to know in which of the PRF evaluations $\mathcal{A}$ uses its real inputs. Thus, $\mathcal{S}$ defines $\mathcal{A}$'s inputs to $\pi_\cap$ to be its inputs to the α^{th} series of oblivious PRF evaluations. In addition to the above, $\mathcal{S}$'s knowledge of α enables it to prepare replies to the checks in Step 6 that $\mathcal{A}$ would expect to receive from an honest P_1. Details follow,

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 computing $\mathcal{F}_{\text{OT}}$:
   (a) If this input is abort_2 or $\mathsf{corrupted}_2$, then $\mathcal{S}$ sends abort_2 or $\mathsf{corrupted}_2$ (respectively) to the trusted party computing $\mathcal{F}_\cap$, simulates P_1 aborting and halts (outputting whatever $\mathcal{A}$ outputs).
   (b) If the input is cheat_2, then $\mathcal{S}$ sends cheat_2 to the trusted party computing $\mathcal{F}_\cap$. If it receives back $\mathsf{corrupted}_2$, then it hands $\mathcal{A}$ the message $\mathsf{corrupted}_2$ as if it received it from the trusted party, simulates P_1 aborting and halts (outputting whatever $\mathcal{A}$ outputs). If it receives back $\mathsf{undetected}$ (together with the input X of the honest P_1), then it proceeds as follows. First, at this point $\mathcal{A}$ expects to receive a pair (s_0, s_1) and $\mathcal{A}$ gives it a pair of random keys $s_0, s_1 \leftarrow \mathcal{I}_{\text{PRP}}(1^{p(n)})$. Next, it uses the input X of P_1 that it obtained in order to perfectly emulate P_1 in the rest of the execution. $\mathcal{S}$ then sends the output that P_1 receives from this execution with $\mathcal{A}$ to the trusted party computing $\mathcal{F}_\cap$ to be the output of the honest party P_1 in the ideal execution. The simulation ends here in this case.

[5] The proof here relies crucially on the fact that $\mathcal{A}$ commits to the values $C_{\text{PRP}_0}, C_{\text{PRP}_1}$ before the keys k_0, k_1 for the pseudorandom functions are determined, and furthermore that these keys are uniformly chosen via coin tossing and not determined by P_1.

(c) 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. In addition, $\mathcal{S}$ 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 computing $\mathcal{F}_{\text{PRF}}$ 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 computing $\mathcal{F}_{\text{PRF}}$, 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 $V_\alpha = \{F_{\text{PRF}}(k_\alpha, F_{\text{PRP}}(s_\alpha, \zeta))\}_{\zeta \in Z}$ 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 $c_{\rho_{1-\alpha}}$. In case $\mathcal{A}$ did not send s_α, $\mathcal{S}$ halts. (Note that there is a possibility that $\mathcal{A}$ will send $s_{1-\alpha}$ instead of s_α which would not cause P_1 to halt in a real execution. However, in the hybrid model $s_{1-\alpha}$ only appears in $C_{\text{PRP}_{1-\alpha}}$ and in $V_{1-\alpha}$ from which $s_{1-\alpha}$ cannot be determined except with negligible probability; this follows from an easy reduction to the security of the pseudorandom permutation.)

8. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Let $\epsilon = \frac{1}{2}$. We prove that

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

where the ensembles are indexed by $X \subseteq \{0,1\}^{p(n)}$ of size m_1, $Y \subseteq \{0,1\}^{p(n)}$ of size m_2, $z \in \{0,1\}^*$ and $n \in \mathbb{N}$.

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 $C_{\mathrm{PRF}_{1-\alpha}}$ (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 $\mathcal{F}_{\mathrm{OT}}$ or $\mathcal{F}_{\mathrm{CT}}$. Due to the similarity to the case where P_1 is corrupted we omit the details here. Let bad denote the event that $\mathcal{A}$ sends abort_2, $\mathsf{corrupted}_2$ or cheat_2. Then relying on the above discussion it holds that,

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

Next we analyze the security in case $\mathcal{A}$ provides valid inputs to $\mathcal{F}_{\mathrm{OT}}$ and $\mathcal{F}_{\mathrm{CT}}$, and prove through a series of games that the output distributions are computationally indistinguishable. In game H_i, we denote the simulator by $\mathcal{S}_i$.

Game H_1**:** In the first game the simulator $\mathcal{S}_1$ has access to an oracle $\mathcal{O}_{F_{\mathrm{PRP}}}$ for computing F_{PRP} and instead of computing F_{PRP} using $s_{1-\alpha}$, it queries the oracle. In contrast, the computation using s_α remains the same. Recall that $\mathcal{S}_1$ does not make any use of the actual key $s_{1-\alpha}$ at any stage of the protocol, and so an oracle can easily be used. We stress that the execution in this game still involves a trusted party that computes $\mathcal{F}_\cap$. Now, 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 $z \in \{0,1\}^*$ let

$$\{\mathrm{H1}_{\mathcal{S}(z)}(X,Y,n)\}_{n\in N}$$

denote the output distribution of $\mathcal{S}$ in this game. Clearly the output distribution of the current and original simulation are identical.

Game H_2**:** In this game we define $\mathcal{S}_2$ who is the same as $\mathcal{S}_1$ except that it uses an oracle $\mathcal{O}_{H_{\mathrm{Perm}}}$ computing a truly random permutation instead of $\mathcal{O}_{F_{\mathrm{PRP}}}$, while the rest of the execution is as above. Then 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 $z \in \{0,1\}^*$ let

$$\{\mathrm{H2}_{\mathcal{S}(z)}(X,Y,n)\}_{n\in N}$$

denote the output distribution of $\mathcal{A}$ in this game. We prove the following:

Claim 3.4 $\{\mathrm{H1}_{\mathcal{S}(z)}(X,Y,n)\}_{n\in N} \stackrel{\mathrm{c}}{\equiv} \{\mathrm{H2}_{\mathcal{S}(z)}(X,Y,n)\}_{n\in N}$

Proof: The proof follows from the security of F_{PRP}. Namely, we show that a distinguisher that distinguishes the above distributions can be translated into a distinguisher for F_{PRP}. Assume that there exist an adversary $\mathcal{A}$, a distinguisher D and infinitely many inputs $(1^n, X)$ and $(1^n, Y)$, such that D distinguishes $\mathcal{A}$'s output in the above games whenever the inputs are $(1^n, X)$ and $(1^n, Y)$. Then a distinguisher D_{PRP} with oracle access to either $\mathcal{O}_{F_{\mathrm{PRP}}}$ or $\mathcal{O}_{H_{\mathrm{Perm}}}$, and an auxiliary input (X, Y, z) is constructed the following way (note that D_{PRP} is given X in order for it to be able to compute $X \cap \tilde{Y}$, where $\tilde{Y}$ is the input of $\mathcal{A}$ as extracted by the simulator). On input 1^n, D_{PRP} invokes $\mathcal{A}(1^n, Y, z)$ and plays the roles of the simulator and the trusted party computing $\mathcal{F}_\cap$. However, whenever D_{PRP} is required to carry out a computation using $s_{1-\alpha}$, it forwards the

evaluated values to its oracle instead and continues with the oracle's answer. Clearly, if D_{PRP}'s oracle computes $\mathcal{O}_{F_{\text{PRP}}}$, then the execution is identical to the execution in game H_1, whereas if the oracle computes $\mathcal{O}_{H_{\text{Perm}}}$ then it is identical to H_2. Thus D_{PRP} distinguishes between F_{PRP} and H_{Perm} with the same probability that D distinguishes between H_1 and H_2. ■

Game H_3**:** The next game is identical to the previous one except that the simulator $\mathcal{S}_3$ 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). Note that $\mathcal{S}_3$ still uses a trusted party that computes $\mathcal{F}_{\cap}$.

Game H_4**:** In this game the simulator $\mathcal{S}_4$, that is still given access to a trusted party computing $\mathcal{F}_{\cap}$, 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. Note that the simulator (invoking $\mathcal{S}_{\text{PRF}}$) extracts $\mathcal{A}$'s input to the αth set of oblivious PRF evaluations, and merely forwards these values to its oracle. It additionally forwards $\mathcal{A}$ the oracle's responses on the set $\{F_{\text{PRP}}(s_\alpha, \zeta)\}_{\zeta \in Z}$ in step 6 (recall that the simulator knows s_α). In both H_3 and H_4 the distribution generated by the pseudorandom function is identical. 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 $\mathcal{O}_{H_{\text{Func}}}$. Using a similar reduction as above, we have that the output distribution of $\mathcal{A}$ in this current game is computationally indistinguishable from its output distribution of the previous game.

Game H_6**:** In this game, the simulator $\mathcal{S}_6$ computes the commitments in C_{PRP_α} using the real input set X of P_1 instead of using random values. Since these commitments are never opened, the indistinguishability of this game to the previous one follows from the hiding property of the commitments.

Game H_7**:** In this game, the simulator $\mathcal{S}_6$ queries its PRF oracle on the real input set X of P_1 in order to construct V_α. That is, $\mathcal{S}$ uses X instead of constructing V_α using the output received by the trusted party computing $\mathcal{F}_{\cap}$ and adding random values (thus in H_5 the simulator uses distinct random values to complete the set Z_T while here it uses $X \setminus X \cap Y$). Note that the only difference from the previous game is regarding the values in V_α that are not included in $X \cap Y$; see step 6 of the simulation. Now, since $\mathcal{O}_{H_{\text{Func}}}$ is a truly random function, we achieve the same output distribution in both games.

Game H_8**:** 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. Using a similar reduction as above we have that H_7 and H_8 are computationally indistinguishable.

Game H_9**:** In this game we modify $\mathcal{O}_{H_{\text{Func}}}$ back into $\mathcal{O}_{F_{\text{PRF}}}$. This replacement affects the αth set of PRF evaluations.

Game H_{10}**:** Finally, we let the simulator $\mathcal{S}_{10}$ to perfectly emulate the role of P_1. In particular, $\mathcal{S}_{10}$ carries out the PRF and PRP computations by itself, and uses k_α as generated by ρ_α committed to in c_{ρ_α}. 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}.

We summarize the steps of the proof in the following table:

Game	Change from previous game	Indistinguishability argument
H_1	PRF using $s_{1-\alpha}$ replaced with $\mathcal{O}_{F_{\text{PRP}}}$	Identical to simulation
H_2	Replace $\mathcal{O}_{F_{\text{PRP}}}$ with $\mathcal{O}_{H_{\text{Perm}}}$	Pseudorandomness of F_{PRP}
H_3	Computation of $V_{1-\alpha}$ and $C_{\text{PRP}_{1-\alpha}}$ uses real input X of P_1 and not random values	Identical to H_3 because truly random permutation is used
H_4	Replace $F_{\text{PRF}}(k_\alpha, \cdot)$ with $\mathcal{O}_{F_{\text{PRF}}}$ using a random key	Based on hiding property of commitment $c_{\rho_\alpha} = \mathsf{com}(\rho_\alpha; r_\alpha)$
H_5	Replace $\mathcal{O}_{F_{\text{PRF}}}$ with $\mathcal{O}_{H_{\text{Func}}}$	Pseudorandomness of F_{PRF}
H_6	Use real input X for C_{PRP_α}	Hiding property of commitments
H_7	Use real input X for V_α	Identical because H_{Func} is random
H_8	Replace $\mathcal{O}_{H_{\text{Perm}}}$ with $\mathcal{O}_{F_{\text{PRP}}}$	As in H_2
H_9	Replace $\mathcal{O}_{H_{\text{Func}}}$ with $\mathcal{O}_{F_{\text{PRF}}}$	As in H_5
H_{10}	Replace $\mathcal{O}_{F_{\text{PRF}}}$ using random key with F_{PRF} using k_α	Hiding property of commitment c_{ρ_α} to ρ_α (used to generate k_α)

Noting again that H_1 is identical to the simulation by $\mathcal{S}$ and that H_{10} is identical to the real execution, we conclude that the ideal simulation by $\mathcal{S}$ is computationally indistinguishable from a 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. 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 can be run in parallel.

4 Secure Pattern Matching

The basic problem of *pattern matching* is the following one: given a text T of length N (for simplicity we assume that N is a power of 2) 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 [4, 19].

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 [10]), 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 $T_i, T_{i+1}, ...$ are *dependent* on each other (e.g. 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 simulation based 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 (see section 2.2.2), in order to have P_2 obtain all of these values while running only a *single* execution of π_{PRF}. We therefore consider a modified version of π_{PRF} for computing the Naor-Reingold function such that P_2's output is the *set* $\{F_{\text{PRF}}(k, p \| \langle i \rangle)\}_{i=1}^{N-m+1}$, rather than just the single value $F_{\text{PRF}}(k, p)$. The corresponding functionality, denoted by $\mathcal{F}_{\text{MPRF}}$, is defined by

$$((k, 1^N), p) \mapsto (\lambda, \{F_{\text{PRF}}(k, p \| \langle i \rangle)\}_{i=1}^{N-m+1})$$

This modification can be achieved as follows. P_1 is given a PRF key $k = (p, q, g^{a_0}, a_1, \ldots, a_{m+\log N})$ with $m + \log N$. Recall that in π_{PRF} the last message received by P_2 is $\tilde{g} = g^{a_0 \cdot \prod_{i=1}^{n} \frac{1}{r_i}}$. Then instead of this P_1 computes and sends the set

$$\left\{ \left(i, g_i = \tilde{g}^{\Pi_{j=1}^{\log N} a_{m+j}^{\langle i \rangle_j}} \right) \right\}_{i=1}^{N-m+1}$$

where $\langle i \rangle_j$ denotes the jth bit in $\langle i \rangle$. Finally, P_2 completes its run as in the original execution of π_{PRF} but relative to every element from the above set, yielding the set $\{f_i = F_{\text{PRF}}(k, p || \langle i \rangle\}_{i=1}^{N-m+1}$. Formally,

Protocol π_{MPRF}

- **Inputs:** The input of P_1 is $k = (p, q, g^{a_0}, a_1, \ldots, a_{n+\log N})$ and a value 1^N, and the input of P_2 is a value x of length n.

- **Auxiliary inputs:** Both parties have the security parameter 1^n and are given the primes p and q.
- **The protocol:**
  1. P_1 chooses n random values $r_1, \ldots, r_n \in_R \mathsf{Z}_q^*$.
  2. The parties engage in a 1-out-2 multi-oblivious transfer protocol π_{OT}^m (with $m = n$ executions). In the ith iteration, P_1 inputs $y_0^i = r_i$ and $y_1^i = r_i \cdot a_i$ (with multiplication in Z_q^*), and P_2 enters the bit $\sigma_i = x_i$ where $x = x_1, \ldots, x_n$. If the output of any of the oblivious transfers is $\perp$, then both parties output $\perp$ and halt. Otherwise:
  3. P_2's output from the n executions is a series of values $y_{x_1}^1, \ldots, y_{x_n}^n$. If any value $y_{x_i}^i$ is not in Z_q^*, then P_2 redefines it to equal 1.
  4. P_1 sets $\tilde{g} = g^{a_0 \cdot \prod_{i=1}^n \frac{1}{r_i}}$ and sends P_2 the set
  $$\left\{ \left(i, g_i = \tilde{g}^{\prod_{j=1}^{\log N} a_{m+j}^{\langle i \rangle_j}} \right) \right\}_{i=1}^{N-m+1}$$
  5. P_2 aborts if the order of any g_i is different than q. Otherwise, P_2 computes and outputs the set
  $$\left\{ \left(i, y_i = g_i^{\prod_{i=1}^n y_{x_i}^i} \right) \right\}_{i=1}^{N-m+1} = \{(i, F_{\text{PRF}}(k, p \| \langle i \rangle))\}_{i=1}^{N-m+1}$$

We continue with the security proof of π_{MPRF}. Due to the similarity to the proof of π_{PRF} we present a proof sketch only.

Proposition 4.1 *Assume that π_{OT}^m securely computes the multi-oblivious transfer functionality with one-sided simulation. Then π_{MPRF} securely computes $\mathcal{F}_{\text{MPRF}}$ with one-sided simulation.*

Proof Sketch: Note first that the only changes in π_{MPRF} are relative to P_1 and thus the security argument for the case that P_1 is corrupted is as in the proof of Proposition 2.4.

As for the case that P_2 is corrupted, we present the proof in the π_{OT}^m-hybrid model. Let $\mathcal{A}$ denote the adversary that controls P_2, then construct a simulator $\mathcal{S}$ as follows. $\mathcal{S}$ receives $\mathcal{A}$'s input x' for the multi-oblivious transfer execution, sends it to the trusted party computing $\mathcal{F}_{\text{PRF}}$ and receives its answer, the set $Z = \{z_i\}_{i=1}^{N-m+1}$. Then, in each oblivious transfer $\mathcal{S}$ hands $\mathcal{A}$ a random value $r_i \in_R \mathsf{Z}_q^*$. Next, $\mathcal{S}$ sets $\tilde{z}_i = z_i^{\prod_{i=1}^n \frac{1}{r_i}}$ for every $z_i \in Z$ and sends (i, z_i) to $\mathcal{A}$. This completes the simulation.

The proof that $\mathcal{A}$'s view is identical in both the simulated and hybrid executions and that $\mathcal{A}$ returns the same value in both executions is as in the proof of Proposition 2.4. This is because the only difference between the executions is with respect to the PRF computation of the set $\{\langle i \rangle\}_{i=1}^{N-m+1}$. Briefly, all the r_i values are distributed identically to the messages sent in the hybrid execution, and given the set Z and all the r_i values, it is easy to define a valid random PRF key (using the original values of $\{a_{n+1}, \ldots, a_{\log N}\}$ that were sent to the trusted party by P_1). Therefore the exact same proof can be applied here as well. ■

Efficiency. Note first that the round efficiency of π_{MPRF} has not been changed with respect to π_{PRF} and is constant. In addition, the number of exponentiations in the batch OT is $14n + 14$. Thus the total number of exponentiations is $14n + 14 + 2N$ (where the additional $2N$ exponentiations are required to compute the set of N values).

The Protocol

We are now ready to present our main result for this section.

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 π_{MPRF} where P_1 enters the key k and P_2 enters its pattern p of length m. The output of P_2 from this execution is the set $\{(i, f_i)\}_{i=1}^{N-m+1}$.
  3. For every i, let $t_i = F_{\mathrm{PRF}}(k, T_i \| \langle i \rangle)$. 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 4.2 *Let F_{PRF} denote the Naor-Reingold function and assume that it is pseudorandom. Furthermore, assume that protocol π_{PRF} securely computes the functionality $((k, 1^N), p) \mapsto (\lambda, \{F_{\mathrm{PRF}}(k, p \| \langle i \rangle)\}_{i=1}^{N-m+1})$ with one-sided simulation. Then protocol π_{PM} securely computes $\mathcal{F}_{\mathrm{PM}}$ with one-sided simulation.*

Proof: We separately consider the case that P_1 and P_2 is corrupted.

Party P_1 is corrupted. Since we are only proving one-sided simulatability here, all we need to show is that P_1 learns nothing about P_2's input. Now, due to the fact that P_1's view only includes messages within π_{MPRF}, by the security of π_{MPRF} with one-sided simulatability we have that P_1 learns nothing about P_2's input p. Beyond that P_1 does not receive any message.

Party P_2 is corrupted. The motivation has been discussed above and we therefore proceed directly to the proof. We present the proof here in the $\mathcal{F}_{\mathrm{MPRF}}$-hybrid model. Let $\mathcal{A}$ denote an adversary controlling P_2. Then we construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ receives p and z and invokes $\mathcal{A}$ on this input.
2. $\mathcal{S}$ receives from $\mathcal{A}$ its input p' as handed to $\mathcal{F}_{\mathrm{MPRF}}$, sends it to the trusted party and receives back the set of text locations I for which there exists a match.
3. $\mathcal{S}$ chooses a key $k \leftarrow \mathcal{I}_{\mathrm{PRF}}(1^m + \log N)$ and sends $\mathcal{A}$ the set $\{(i, F_{\mathrm{PRF}}(k, p' \| \langle i \rangle))\}_{i=1}^{N-m+1}$, as the trusted party that computes $\mathcal{F}_{\mathrm{MPRF}}$ would.
4. Let $k = (p, q, g^{a_0}, a_1, \ldots, a_{m+\log N})$. Then for every $i \in 1 \leq i \leq N - m + 1$, $\mathcal{S}$ continues as follows:
   - If $i \in I$, the simulator $\mathcal{S}$ defines $t_i = F_{\mathrm{PRF}}(k, p' \| \langle i \rangle)$.
   - Otherwise, $\mathcal{S}$ defines $t_i = F_{\mathrm{PRF}}(k, \hat{p} \| \langle i \rangle)$ where $\hat{p} \neq p'$ is an arbitrary string.
5. $\mathcal{S}$ hands $\mathcal{A}$ the set $\{(i, t_i)\}$ and outputs whatever $\mathcal{A}$ outputs.

Note that the only difference between the real and the simulated executions is in the last step where, for every text location i such that $T_i \neq p'$, $\mathcal{S}$ defines t_i based on a fixed $\hat{p} \neq p'$ instead of basing it on the substring T_i (which is unknown to the simulator). Intuitively, this makes no difference by the pseudorandomness of the function. Formally, we have the following steps:

Game H_1**:** We begin by modifying $\mathcal{S}$ so that it uses an oracle $\mathcal{O}_{F_{\mathrm{PRF}}}$ for computing the function $\mathcal{F}_{\mathrm{PRF}}$. $\mathcal{S}$ can use this oracle by handing $\mathcal{A}$ the set $F = \{\mathcal{O}_{F_{\mathrm{PRF}}}(p' \| \langle i \rangle)\}_{i=0}^{N-m+1}$ as its output from the trusted party computing $\mathcal{F}_{\mathrm{MPRF}}$. Furthermore, it can define $t_i = \mathcal{O}_{F_{\mathrm{PRF}}}(p' \| \langle i \rangle)$ if $i \in I$, and $t_i = \mathcal{O}_{F_{\mathrm{PRF}}}(\hat{p} \| \langle i \rangle)$ otherwise. By definition of $\mathcal{F}_{\mathrm{PRF}}$, this is exactly the same distribution as generated by $\mathcal{S}$ above. We stress that a trusted party that computes $\mathcal{F}_{\mathrm{PM}}$ still involves in this game.

Game H_2**:** Next, we replace $\mathcal{O}_{F_{\mathrm{PRF}}}$ with an oracle $\mathcal{O}_{H_{\mathrm{Func}}}$ computing a truly random function. Clearly, the resulting distributions in both games are computationally indistinguishable. This can be proven via a reduction to the pseudorandomness of the function F_{PRF}. Informally, let D_{PRF} denote a distinguisher who attempts to distinguish F_{PRF} from H_{Func}. Then D_{PRF}, playing the role of $\mathcal{S}$ above, invokes its oracle on the sets $\{p_i = p' \| \langle i \rangle\}_{i=0}^{N-m+1}$ and $\{t_i\}_{i=0}^{N-m+1}$, where $t_i = p_i$ when $i \in I$, and $t_i = \hat{p} \| \langle i \rangle$ otherwise (note that the difference is if p' or $\hat{p}$ is used). Now, any distinguisher for the distributions of games H_1 and H_2 can be utilized by D_{PRF} to distinguish between F_{PRF} and H_{Func}.

Game H_3**:** We now modify $\mathcal{S}$ so that it computes all of the t_i values correctly using the honest P_1's text T instead of invoking a trusted party. The resulting distribution is identical because the oracle computes a truly random function and all inputs are distinct in both cases (the distinction is due to the index i that is concatenated each time).

Game H_4**:** Next, we modify the oracle $\mathcal{O}_{H_{\mathrm{Func}}}$ back to an oracle $\mathcal{O}_{\mathrm{PRF}}$ computing F_{PRF}. Again, the distributions in games H_3 and H_4 are computationally indistinguishable by a straightforward reduction.

Game H_5**:** Finally, we compute the pseudorandom function instead of using an oracle. This makes no difference whatsoever for the output distribution.

Noting that the last game is exactly the distribution generated in a hybrid execution, we have that the hybrid and ideal executions are computationally indistinguishable, completing the proof. ■

Efficiency. As for every protocol presented here, π_{PM} has a constant number of rounds. In addition, the number of exponentiations computed is $14m + 14 + 3N$.

One-sided versus full simulatability. Observe that our protocol does not achieve correctness when P_1 is corrupted because P_1 may construct the t_i values in a way that is not consistent with any text T. Specifically, for every i, the last $m-1$ bits of T_i are supposed to be the first $m-1$ bits of T_{i+1}, but P_1 is not forced to construct the values in this way. Our protocol is therefore not simulatable in this case (even when considering only covert adversaries), and we do not know how to enforce such behavior efficiently.

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. Agrawal and R. Srikant. Privacy-Preserving Data Mining. In the *2000 SIGMOD Conference*, pages 439450, 2000.

[4] R.S. Boyer, and J.S. Moore. A Fast String Searching Algorithm. *Communications of the Association for Computing Machinery*, 20:762–772, 1977.

[5] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[6] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[7] R. Cleve. Limits on the Security of Coin Flips when Half the Processors are Faulty. In 18*th STOC*, pages 364–369, 1986.

[8] 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.

[9] U. Feige and A. Shamir. Zero Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO'89,* Springer-Verlag (LNCS 435), pages 526–544, 1989.

[10] 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.

[11] 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.

[12] O. Goldreich. *Foundations of Cryptography: Volume 1 - Basic tools*. Cambridge University Press, 2001.

[13] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge University Press, 2004.

[14] 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.

[15] 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.

[16] C. Hazay and Y. Lindell. Efficient Oblivious Polynomial Evaluation and Transfer with Simulation-Based Security. Manuscript, 2008.

[17] J. Katz. Bridging Game Theory and Cryptography: Recent Results and Future Directions. In the 5*th TCC*, Springer-Verlag (LNCS 4948), pages 251–272, 2008.

[18] L. Kissner and D.X. Song. Privacy-Preserving Set Operations. In *CRYPTO 2005*, Springer-Verlag (LNCS 3621), pages 241–257, 2005.

[19] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6:323–350, 1977.

[20] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology*, 16(3):143–184, 2003.

[21] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[22] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[23] M. Naor and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. In 31*st STOC*, pages 245–254, 1999.

[24] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In 12*th SODA*, pages 448–457, 2001.

[25] M. Naor and O. Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. In 38*th FOCS*, pages 231–262, 1997.

[26] T. P. Pedersen. Non-Interactive and Information-Theoretical Secure Verifiable Secret Sharing. In *CRYPTO 1991*, Springer-Verlag (LNCS 576) pp. 129–140, 1991.

[27] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[28] A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.