Scalable Multi-Party Private Set-Intersection

Carmit Hazay* Muthuramakrishnan Venkitasubramaniam†

Abstract

In this work we study the problem of private set-intersection in the multi-party setting and design two protocols with the following improvements compared to prior work. First, our protocols are designed in the so-called star network topology, where a designated party communicates with everyone else, and take a new approach of leveraging the 2PC protocol of [FNP04]. This approach minimizes the usage of a broadcast channel, where our semi-honest protocol does not make any use of such a channel and all communication is via point-to-point channels. In addition, the communication complexity of our protocols scales with the number of parties.

More concretely, (1) our first semi-honest secure protocol implies communication complexity that is linear in the input sizes, namely $O((\sum_{i=1}^{n} m_i) \cdot \kappa)$ bits of communication where κ is the security parameter and m_i is the size of P_i's input set, whereas overall computational overhead is quadratic in the input sizes only for a designated party, and linear for the rest. We further reduce this overhead by employing two types of hashing schemes. (2) Our second protocol is proven secure in the malicious setting. This protocol induces communication complexity $O((n^2 + nm_{\text{MAX}} + nm_{\text{MIN}} \log m_{\text{MAX}})\kappa)$ bits of communication where m_{MIN} (resp. m_{MAX}) is the minimum (resp. maximum) over all input sets sizes and n is the number of parties.

Keywords: Scalable Multi-Party Computation, Private Set-Intersection

1 Introduction

Background on secure multi-party computation. Secure multi-party computation enables a set of parties to mutually run a protocol that computes some function f on their private inputs, while preserving a number of security properties. Two of the most important properties are privacy and correctness. The former implies data confidentiality, namely, nothing leaks by the protocol execution but the computed output. The latter requirement implies that the protocol enforces the integrity of the computations made by the parties, namely, honest parties learn the correct output. Feasibility results are well established [Yao86, GMW87, MR91, Bea91], proving that any efficient functionality can be securely computed under full simulation-based definitions (following the ideal/real paradigm). Security is typically proven with respect to two adversarial models: the semi-honest model (where the adversary follows the instructions of the protocol but tries to learn more than it should from the protocol transcript), and the malicious model

*Bar-Ilan University, Israel. Email: carmit.hazay@biu.ac.il. Supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Ministers Office, and by a grant from the Israel Ministry of Science and Technology (grant No. 3-10883).

†University of Rochester, Rochester, NY 14611, NY. Email: muthuv@cs.rochester.edu. Supported by Google Faculty Research Grant and NSF Awards CNS-1526377 and CNS-1618884. The views expressed are those of the authors and do not reflect the official policy or position of Google, the Department of Defense, the National Science Foundation, or the U.S. Government. .

(where the adversary follows an arbitrary polynomial-time strategy), and feasibility holds in the presence of both types of attacks.

Following these works, many constructions focused on improving the *efficiency* of the computational and communication costs. Conceptually, this line of works can be split into two sub-lines: (**1**) Improved generic protocols that compute any boolean/arithmetic circuit; see [IPS08, LOP11, BDOZ11, DPSZ12, LPSY15] for just a few examples. (**2**) Protocols for concrete functionalities. In the latter approach attention is given to constructing efficient protocols for specific functions while exploiting their internal structure. While this approach has been proven useful for many different two-party functions in both the semi-honest and malicious settings such as calculating the kth ranked element [AMP04], pattern matching and related search problems [HT10, Ver11], set-intersection [JL09, HN12], greedy optimizations [SV15] and oblivious pseudorandom function (PRF) evaluation [FIPR05], only minor progress has been achieved for concrete multi-party functions.

2PC private set-intersection. The set-intersection problem is a fundamental functionality in secure computation and has been widely studied in the past decade. In this problem a set of parties $P_1, \ldots, P_n$, holding input sets $X_1, \ldots, X_n$ of sizes $m_1, \ldots, m_n$, respectively, wish to compute $X_1 \cap X_2 \cap \ldots \cap X_n$. In the two-party setting this problem has been intensively studied by researchers in the last few years mainly due to its potential applications for dating services, datamining, recommendation systems, law enforcement and more, culminating with highly efficient protocols with practically linear overhead in the set sizes; see for instance [FNP04, DSMRY09, JL09, HL10, HN12, Haz15]. For example, consider two security agencies that wish to compare their lists of suspects without revealing their contents, or an airline company that would like to check its list of passengers against the list of people that are not allowed to go abroad.

Two common approaches are known to concretely solve this problem securely in the plain model for two parties: (**1**) oblivious polynomial evaluation (OPE) and (**2**) committed oblivious PRF evaluation.

In the first approach based on OPE, one party, say P_1, computes a polynomial $Q(\cdot)$ such that $Q(x) = 0$ for all $x \in X_1$. The set of coefficients of $Q(\cdot)$ are then encrypted using a homomorphic encryption scheme and sent to the other party P_2, who then computes the encryption of $r_{x'} \cdot Q(x') + x'$ for all $x' \in X_2$ using fresh randomness $r_{x'}$ via homomorphic evaluation. Finally, P_1 decrypts these computed ciphertexts and outputs the intersection of its input set X_1 and these plaintexts. This is the approach (and variants thereof) taken by the works [FNP04, DSMRY09, HN12].

The second approach uses a secure implementation of oblivious PRF evaluation. More precisely, in this approach, party P_1 chooses a PRF key K and computes the set $\mathsf{PRF}_{X_1} = \{\mathsf{PRF}_K(x)\}_{x \in X_1}$. The parties then execute an oblivious PRF protocol where P_1 inputs the key K and P_2 inputs its private set X_2. At the end of this protocol P_2 learns the set $\mathsf{PRF}_{X_2} = \{\mathsf{PRF}_K(x')\}_{x' \in X_2}$. Finally, P_1 sends the set PRF_{X_1} to P_2, and P_2 computes $S = \mathsf{PRF}_{X_1} \cap \mathsf{PRF}_{X_2}$ and outputs the corresponding elements $x' \in X_2$ whose PRF values are in S as the actual intersection. This idea was introduced in [FIPR05] and further used in [HL10, JL09, JL10].

Recent significant concrete improvements were shown in [PSZ14, PSZ18] in the semi-honest model based on oblivious-transfer (OT) extension and Bloom filters [KKRT16], and in [RR17] in the malicious setting based on Bloom filters. Other solutions in the random oracle model such as [CT10, CKT10, ACT11] take a different approach by applying the random oracle on (one of) the sets members, or apply oblivious transfer extension [DCW13] to implement a garbled Bloom filter.

By now, major progress had already been achieved for general two-party protocols [KSS12, FJN+13, GLNP15, Lin16]. Moreover, it has been surprisingly demonstrated that general protocols can be more efficient than the concrete "custom-made" protocols for set-intersection [HEK12].

MPC private set-intersection. While much progress has been made towards achieving practical protocols in the two-party setting to realize set-intersection, only few works have considered so far the multi-party setting. Moreover, most of the previous approaches fail to leverage the highly efficient techniques that were developed for the two-party case with scalable efficiency. Specifically, while several recent works improve the efficiency of generic multi-party protocols [LPSY15, LSS16, KOS16], they still remain inefficient for concrete applications on big data due to high communication complexity overhead.

The first concrete protocols that securely implemented the set-intersection functionality were designed by Kissner and Song [KS05]. The core technique underlying these protocols is based on OPE and extends the [FNP04] approach, relying on expensive generic zero-knowledge proofs to achieve correctness. Following that, Sang and Shen introduced a new protocol with quadratic overhead in the size of the input sets [SS07], which was followed by another protocol in the honest majority setting based on Bilinear groups [SS08]. Cheon et al. improved the communication complexity of these works by reducing the dependency on the input sets from quadratic to quasi linear [CJS12]. Nevertheless, each party still needs to broadcast $O(m_i)$ elements, where m_i is the size of its input set, implying that the overall communication complexity and group multiplications per player grow quadratically with the number of parties. In [DMRY11], the authors considered a new approach based on multivariate polynomials achieving broadcast communication complexity of $O(n \cdot m_{\text{MAX}} + m_{\text{MAX}} \cdot \log^2 m_{\text{MAX}})$ and computational complexity $O(n \cdot m_{\text{MAX}}^2)$, where m_{MAX} is the maximum over all input sets sizes and n is the number of parties. Finally, in a recent work [MN15], Miyaji and Nishida introduced a semi-honest secure protocol based on Bloom filters that achieves communication complexity $O(n \cdot m_{\text{MAX}})$ and computational complexity $O(n \cdot m_{\text{MAX}})$ for the designated party. This work was further extended in [MNN17] achieving similar bounds.

One can also consider using standard secure computation to securely realize set-intersection. One popular approach for efficient protocols is [DPSZ12] protocol, dubbed SPDZ, that describes a flavour of [GMW87] protocol for arithmetic circuits. This protocol consists of a preprocessing phase that uses somewhat homomorphic encryption scheme to generate correlated randomness, that is later used in an information theoretic online phase. The total overhead of this approach is $O(n \cdot s + n^3)$ where s is the size of the computed circuit. An alternative approach to compute the offline phase, avoiding these costly primitives, was recently introduced in [KOS16]. This protocol achieves a significant improvement, and is only six times less efficient than a semi-honest version of the protocol ((where their experiments were shown for up to five parties), yet its cost still approaches $O(n^2)$ overhead per multiplication triple. Finally, we note that the round complexity of this approach is proportional to the circuit's multiplication depth.

A different approach was taken in [BMR90], extending the celebrated garbled circuits technique of [Yao86] to the multi-party setting. This constant-round protocol, developed by Beaver, Micali and Rogaway, has proven secure in the presence of semi-honest adversaries (and malicious adversaries in the honest majority setting). It is comprised of an offline phase for which the garbled circuit is created, and an online phase for which the garbled circuit is evaluated. Recently, Lindell et al. [LPSY15] extended the [BMR90] protocol to the malicious honest majority setting. For the offline phase the authors presented an instantiation based on [DPSZ12]. In a more recent work, Lindell et al. [LSS16] introduced a concretely efficient MPC protocol with malicious security, focusing on reducing the round complexity into 9 rounds. The efficiency of this approach is dominated by the efficiency of the protocol that realizes the offline phase.

Our main motivation in this paper is to develop a new approach for securely realizing set-intersection in the multi-party setting. Concretely, we study whether the multi-party variant of set-intersection can be reduced to the two-party case. Meaning, can we securely realize private multi-party set-intersection using two-party set-intersection protocols. Generally speaking, the paradigm of constructing multi-party protocols from two-party protocols has several important advantages. First, it may require using a broadcast

channel fewer times than in the classic approach (where every party typically communicates with everyone else all the time). Moreover, it enables to leverage the extensive knowledge and experience gained while studying the two-party variant in order to achieve efficient multi-party protocols. Finally, the mere idea of working on smaller pieces of the inputs/problems also implies that we can achieve better running times and implementations. Our new approach has not been considered yet in the past, specifically because it is quite challenging to use two-party protocols for intermediate computations without violating the privacy of the multi-party construction, and required pursuing a new approach.

In light of this overview we pose the following questions,

> *Can we securely realize the set-intersection functionality with linear communication complexity (and sub-quadratic computational complexity) in the input sets sizes?*

In particular, to what extent can multi-party set-intersection be reduced to its two-party variant. Considering the set-intersection functionality, at first sight, it seems that the answer to this question is negative as any 2PC protocol that operates only on two input sets leaks information about the these intersections, which is more than what should be leaked about the outputs by the protocol. One potential solution would be to split the parties into pairs that repetitively compute their pairwise intersection. While it is not clear how to prevent any leakage within iterations, we further note that the round complexity induced by such an approach is $O(\log n)$ where n is the number of parties, and that the number of 2PC invocations is quadratic. It is worth noting that [CKMZ14] also considered an approach of designing a three parties protocol by emulating a two-party protocol, yet their techniques are quite different.

1.1 Our Results

In this paper we devise new protocols that securely compute the set-intersection functionality in the multi-party setting while exploiting known techniques from the two-party setting. In particular, we are able to save on quadratic overhead in pairwise communication that is incurred in typical multiparty protocols and obtain efficient protocols. More specifically, we consider a different network topology than point-to-point fully connected network for which a single designated party communicates with every party (i.e. star topology). An added benefit of this topology is that not all parties must be online at the same time. This topology has been recently considered in [HLP11] in a different context. In this work we consider both the semi-honest and malicious settings.

The semi-honest setting. The main building block in our design is a threshold additively homomorphic public-key encryption scheme (PKE). Our main observation is that one can employ the 2-round semi-honest variant of the [FNP04] protocol, where a designated party P_1 first interacts individually with every other party via a variant of this protocol and learns the (encrypted) cross intersection with every other party. Then in a second stage, P_1 combines these results and computes the outcome. More specifically, we leverage the following core insight, where any element in P_1's input that appears in all other input sets is part of the set-intersection. On the other hand, if some element from P_1's set does not appear in one of the other sets then surely this element is not part of the set-intersection. Therefore, it is sufficient to only examine P_1's set against the other sets rather than examine all pairwise sets, which is the common approach in prior works. Note that our protocol is the first multi-party protocol for realizing private set-intersection that does not need to employ any broadcast channel at any phase during its execution, since all the communication is conducted directly between P_1 and each other party at a point-to-point level. More formally,

Theorem 1.1 (Informal)**.** *Assume the existence of a threshold additively homomorphic encryption scheme. Then, there exists a protocol that securely realizes the private set-intersection functionality in the presence of semi-honest adversaries with no use of a broadcast channel and for $n \geq 2$ parties.*

Moreover, the communication complexity of our protocol is linear in the input sets sizes, namely, $O((\sum_{i=1}^{n} m_i) \cdot \kappa)$ bits of communication where κ is the security parameter, whereas the computational overhead is quadratic in the input sizes only the designated party P_1, namely $O(m_1^2)$ exponentiations (where the overhead of the rest of the parties is a linear number of exponentiations in their input sets). Consequently, the designated party can be set as the party with the *smallest input set*. Finally, by employing hash functions techniques, as in [FNP04], we can further reduce P_1's overhead by splitting the input elements into bins. We consider two hash schemes: simple hashing and balanced allocation hashing. For simple hashing, this approach induces $O((n-1) \cdot m_{\text{MIN}} \cdot \log m_{\text{MAX}})$ overhead where m_{MIN} (resp. m_{MAX}) is the minimum (resp. maximum) over all input sets sizes and n is the number of parties. Whereas for balanced allocation hash functions this approach induces $O((n-1) \cdot m_{\text{MIN}} \cdot \log \log m_{\text{MAX}})$ overhead. In both cases the communication complexity is $O(\mathcal{B} \cdot M \cdot (n-1))$ where $\mathcal{B}$ is the number of bins and M is the maximum bin size.

We note that the first variant based on simple hashing induces a simpler protocol and the modification compared to the original protocol are minor. On the other hand, the protocol based on balanced allocation hashing is slightly more complicated as this hashing, that uses two hash functions, implies two oblivious polynomial evaluations per elements from P_1's input. Consequently, P_1 must somehow learn which of the evaluations (if any) has evaluated to zero. We solve this issue in two ways: either the parties communicate and compute the product of the two evaluations, or the underlying additively homomorphic encryption scheme allows the evaluation degree-2 polynomials (e.g., [BGN05, CF15]). Finally, we note that our approach is the first to employ these techniques due to its internal design that heavily relies on a 2PC approach.

The malicious setting. Next, we extend our semi-honest approach for the malicious setting. In this setting we need to work harder in order to ensure correctness since a corrupted P_1 can easily cheat, by using different input sets in the 2PC executions against different parties. It is therefore crucial that P_1 first broadcasts its committed input to the rest of the parties. Where later, each 2PC protocol is carried out with respect to these commitments. It turns out that even by adding this broadcast phase it is not enough to boost the security of our semi-honest protocol since P_1 may still abuse the security of the [FNP04] protocol. Specifically, the main challenge is to prevent P_1 from learning additional information about the intersection with individual parties as a corrupted P_1 may use ill formed ciphertexts or ciphertexts for which it does not know their corresponding plaintexts, exploiting the honest parties as a decryption oracle.

We recall that the [FNP04] follows by having the parties send encryptions of polynomials defined by their input sets (as explained above). Then, towards achieving malicious security, we design a polynomial check that verifies that P_1 indeed assembled the encrypted polynomials correctly. This check follows by asking the parties to sample a random element u which they later evaluate their encrypted polynomials on and then compare these outcomes against the evaluation of the combined protocol (which is publicly known). To avoid malleability issues, we enforce correctness using a non-malleable proof of knowledge that is provided by each party relative to its computation. This crucial phase allows the simulator to extract the parties' inputs by rewinding them on distinct random values. Interestingly, this proof is only invoked once and thus induces an overhead that is independent of the set sizes. We prove the following theorem.

Theorem 1.2 (Informal)**.** *Assume the existence of a threshold additively homomorphic encryption scheme and simulation sound zero-knowledge proof of knowledge. Then, there exists a protocol that securely realizes the private set-intersection functionality in the presence of malicious adversaries and for $n \geq 2$ parties.*

The communication complexity of the maliciously secure protocol is bounded by $O((n^2 + nm_{\text{MAX}} + nm_{\text{MIN}} \cdot \log m_{\text{MAX}})\kappa)$ bits of communication where m_{MIN} (resp. m_{MAX}) is the minimum (resp. maximum) over all input sets sizes and n is the number of parties. The significant term in this complexity is $O(n \cdot m_{\text{MAX}} \cdot \kappa)$ and this is linearly dependent on both the number of parties and the database size. In contrast, previous works required higher complexity [DMRY11, CJS12]. In terms of of computational overhead, except for party P_1, the computational complexity of each party P_i is $O(m_{\text{MAX}})$ exponentiations plus $O(m_{\text{MIN}} \cdot m_{\text{MAX}})$ groups multiplications, whereas party P_1 needs to perform $O(m_1 \cdot m_{\text{MAX}})$ exponentiations.

Finally, we note that our building blocks can be instantiated based on the El Gamal [Gam85] or Paillier [Pai99] public key encryptions schemes for the semi-honest protocol. In the malicious setting, we either consider the El Gamal scheme together with a Σ-protocol zero-knowledge proof of knowledge, that can be made non-interactive using the FiatShamir heuristic [FS86] which is analyzed in the Random Oracle Model of Bellare and Rogaway [BR93]. The analysis in this model implies the simulation soundness property we need for non-malleability. A second instantiation can be shown based on the [BBS04] public key encryption scheme and the simulation-sound non-interactive zero-knowledge (NIZK) by Groth [Gro06].

2 Preliminaries

2.1 Basic Notations

We denote the security parameter by κ. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. We use the abbreviation PPT to denote probabilistic polynomial-time. We further denote by $a \leftarrow A$ the random sampling of a from a distribution A, by $[d]$ the set of elements $(1, \ldots, d)$ and by $[0, d]$ the set of elements $(0, \ldots, d)$.

We now specify the definition of computationally indistinguishable.

Definition 2.1. *Let $X = \{X(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ be two distribution ensembles. We say that X and Y are* computationally indistinguishable*, denoted $X \overset{c}{\approx} Y$, if for every PPT machine D, every $a \in \{0,1\}^*$, every positive polynomial $p(\cdot)$ and all sufficiently large κ:*

$$\left|\Pr\left[D(X(a,\kappa), 1^\kappa) = 1\right] - \Pr\left[D(Y(a,\kappa), 1^\kappa) = 1\right]\right| < \frac{1}{p(\kappa)}.$$

We define a d-degree polynomial $Q(\cdot)$ by its set of coefficients $(q_0, \ldots, q_d)$, or simply write $Q(x) = q_0 + q_1 x + \ldots q_d x^d$. Typically, these coefficients will be picked from $\mathbb{Z}_p$ for a prime p. We further write $g^{Q(\cdot)}$ to denote the coefficients of $Q(\cdot)$ in the exponent of a generator g of a multiplicative group $\mathbb{G}$ of prime order p.

2.2 Hardness Assumptions

Let $\mathcal{G}$ be a group generation algorithm, which outputs $(p, \mathbb{G}, \mathbb{G}_1, e, g)$ given 1^κ, where $\mathbb{G}, \mathbb{G}_1$ is the description of groups of prime order p, e is a bilinear mapping (see below) and g is a generator of $\mathbb{G}$.

Definition 2.2 (DLIN)**.** *We say that the* decisional linear *problem is hard relative to $\mathcal{G}$, if for any PPT distinguisher D there exists a negligible function* negl *such that*

$$(p, \mathbb{G}, \mathbb{G}_1, e, g, g^x, g^y, g^{xr}, g^{ys}, g^{r+s}) \approx_c (p, \mathbb{G}, \mathbb{G}_1, e, g, g^x, g^y, g^{xr}, g^{ys}, g^d)$$

where $(p, \mathbb{G}, \mathbb{G}_1, e, g) \leftarrow \mathcal{G}(1^\kappa)$ and $x, y, r, s, d \leftarrow \mathbb{Z}_p$.

Definition 2.3 (DDH)**.** *We say that* the decisional Diffie-Hellman (DDH) problem is hard relative to $\mathcal{G}$*, if for any PPT distinguisher D there exists a negligible function* negl *such that*

$$\left| \Pr\left[D(\mathbb{G}, p, g, g^x, g^y, g^z) = 1\right] - \Pr\left[D(\mathbb{G}, p, g, g^x, g^y, g^{xy}) = 1\right] \right| \leq \mathsf{negl}(\kappa),$$

where $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\kappa)$ and the probabilities are taken over the choices of $x, y, z \leftarrow_R \mathbb{Z}_p$.

Definition 2.4 (Bilinear pairing)**.** *Let $\mathbb{G}$, $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p and let g be a generator of $\mathbb{G}$. A map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map for $\mathbb{G}$ if it has the following properties:*

1. *Bi-linearity: $\forall u, v \in \mathbb{G}$, $\forall a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.*
2. *Non-degeneracy: $e(g, g)$ generates $\mathbb{G}_T$.*
3. *e is efficiently computable.*

We assume that the D-linear assumption holds in $\mathbb{G}$.

2.3 Public Key Encryption Schemes (PKE)

We specify first the definitions of public key encryption and IND-CPA.

Definition 2.5 (PKE)**.** *We say that $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a* public key encryption scheme *if* $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ *are polynomial-time algorithms specified as follows:*

- Gen*, given a security parameter 1^κ, outputs keys $(\mathrm{PK}, \mathrm{SK})$, where* PK *is a public key and* SK *is a secret key. We denote this by $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$.*
- Enc*, given the public key* PK *and a plaintext message m, outputs a ciphertext c encrypting m. We denote this by $c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m)$; and when emphasizing the randomness r used for encryption, we denote this by $c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m; r)$.*
- Dec*, given the public key* PK*, secret key* SK *and a ciphertext c, outputs a plaintext message m s.t. there exists randomness r for which $c = \mathsf{Enc}_{\mathrm{PK}}(m; r)$ (or $\perp$ if no such message exists). We denote this by $m \leftarrow \mathsf{Dec}_{\mathrm{PK},\mathrm{SK}}(c)$.*

For a public key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we consider the following *IND-CPA game*:

$(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$.
$(m_0, m_1, history) \leftarrow \mathcal{A}_1(\mathrm{PK})$, s.t. $|m_0| = |m_1|$.
$c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m_b)$, where $b \leftarrow \{0, 1\}$.
$b' \leftarrow \mathcal{A}_2(c, history)$.
$\mathcal{A}$ wins if $b' = b$.

Denote by $\mathrm{ADV}_{\Pi,\mathcal{A}}(\kappa)$ the probability that $\mathcal{A}$ wins the IND-CPA game.

Definition 2.6 (IND-CPA)**.** *A public key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has* indistinguishable encryptions under chosen plaintext attacks *(IND-CPA), if for every non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function* negl *such that $\mathrm{ADV}_{\Pi,\mathcal{A}}(\kappa) \leq \frac{1}{2} + \mathsf{negl}(\kappa)$.*

2.3.1 Additively Homomorphic PKE

A public key encryption scheme is additively homomorphic if given two ciphertexts $c_1 = \mathsf{Enc}_{\mathrm{PK}}(m_1; r_1)$ and $c_2 = \mathsf{Enc}_{\mathrm{PK}}(m_2; r_2)$ it is possible to efficiently compute $\mathsf{Enc}_{\mathrm{PK}}(m_1 + m_2; r)$ with independent r, and without the knowledge of the secret key. Clearly, this assumes that the plaintext message space is a group; we actually assume that both the plaintext and ciphertext spaces are groups (with respective group operations + or $\cdot$). We abuse notation and use $\mathsf{Enc}_{\mathrm{PK}}(m)$ to denote the random variable induced by $\mathsf{Enc}_{\mathrm{PK}}(m; r)$ where r is chosen uniformly at random. We have the following formal definition,

Definition 2.7 (Homomorphic PKE)**.** *We say that a public key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* homomorphic *if for all* k *and all* $(\mathrm{PK}, \mathrm{SK})$ *output by* $\mathsf{Gen}(1^\kappa)$*, it is possible to define groups* $\mathcal{M}, \mathcal{C}$ *such that:*

- *The plaintext space is* $\mathcal{M}$*, and all ciphertexts output by* $\mathsf{Enc}_{\mathrm{PK}}(\cdot)$ *are elements of* $\mathcal{C}$*.*[1]
- *For every* $m_1, m_2 \in \mathcal{M}$ *it holds that*

$$\{\mathrm{PK}, c_1 = \mathsf{Enc}_{\mathrm{PK}}(m_1), c_1 \cdot \mathsf{Enc}_{\mathrm{PK}}(m_2)\} \equiv \{\mathrm{PK}, \mathsf{Enc}_{\mathrm{PK}}(m_1), \mathsf{Enc}_{\mathrm{PK}}(m_1 + m_2)\}$$

  where the group operations are carried out in $\mathcal{C}$ *and* $\mathcal{M}$*, respectively, and the randomness for the distinct ciphertexts are independent.*

Note that any such a scheme supports a multiplication of a plaintext by a scalar. We implicitly assume that each homomorphic operation on a set of ciphertexts is concluded with a refresh operation, where the party multiplies the result ciphertext with an independently generated ciphertext that encrypts zero. This is required in order to ensure that the randomness of the outcome ciphertext is not related to the randomness of the original set of ciphertexts.

2.3.2 Threshold PKE

In a distributed scheme, the parties hold shares of the secret key so that the combined key remains a secret. In order to decrypt, each party uses its share to generate an intermediate computation which are eventually combined into the decrypted plaintext. To formalize this notion, we consider two multi-party functionalities: One for securely generating a secret key while keeping it a secret from all parties, whereas the second functionality jointly decrypts a given ciphertext. We denote the key generation functionality by $\mathcal{F}_{\mathrm{GEN}}$, which is defined as follows,

$$(1^\kappa, \ldots, 1^\kappa) \mapsto \Big((\mathrm{PK}, \mathrm{SK}_1), \ldots, (\mathrm{PK}, \mathrm{SK}_n)\Big)$$

where $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$, and SK_1 through SK_n are random shares of SK. In the simulation, the simulator obtains a public key $\widetilde{\mathrm{PK}}$, either from the trusted party or from the reduction, and enforces that outcome. Namely, that $\mathrm{PK} = \widetilde{\mathrm{PK}}$. Moreover, the decryption functionality $\mathcal{F}_{\mathrm{DEC}}$ is defined by,

$$((c, \mathrm{PK}), \mathrm{PK}, \ldots, \mathrm{PK}) \mapsto \Big((m : c = \mathsf{Enc}_{\mathrm{PK}}(m)), -, \ldots, -\Big).$$

In the simulation, the simulator sends ciphertexts on behalf of the honest parties which do not necessarily match the distribution of ciphertexts in the real execution (as it computes these ciphertexts based on arbitrary

[1] The plaintext and ciphertext spaces may depend on PK; we leave this implicit.

inputs). Moreover, in the reduction the simulator is given a ciphertext (or more) from an external source and must be able to decrypt it, jointly with the rest of the corrupted parties, without knowing the secret key. We therefore require that in the simulation, the simulator cheats in the decryption by biasing the decrypted value into some predefined plaintext $m_{\mathcal{S}}$. This can be formalized by defining the pair of simulators $(\mathcal{S}_{\text{GEN}}, \mathcal{S}_{\text{DEC}})$ as a stateful algorithm where $\mathcal{S}_{\text{DEC}}$ obtains a state returned by $\mathcal{S}_{\text{GEN}}$ which includes the public key enforced by $\mathcal{S}_{\text{GEN}}$ as well as the corrupted parties' shares (but not the decryption key). For simplicity we leave this state implicit. Finally, we consider a variation of $\mathcal{F}_{\text{DEC}}$, denoted by $\mathcal{F}_{\text{DecZero}}$, that allows the parties to learn whether a ciphertext encrypts zero or not, but nothing more. A protocol that realizes $\mathcal{F}_{\text{DecZero}}$ can be easily constructed from any protocol that realizes $\mathcal{F}_{\text{DEC}}$ by having each party raise the encrypted plaintext with a random element and then decrypt the combined ciphertexts, we provide more details below.

These functionalities can be securely realized based on varies PKEs e.g., [Gam85], [BBS04], [Pai99] and [BGN05]. We denote the corresponding protocols that respectively realize $\mathcal{F}_{\text{GEN}}$ and $\mathcal{F}_{\text{DEC}}$ in the semi-honest setting by $\pi^{\text{SH}}_{\text{GEN}}$ and $\pi^{\text{SH}}_{\text{DEC}}$, and by $\pi^{\text{ML}}_{\text{GEN}}$ and $\pi^{\text{ML}}_{\text{DEC}}$ their malicious variants.

2.3.3 The El Gamal PKE

A useful implementation of homomorphic PKE is the El Gamal [Gam85] scheme that has two variations of additive and multiplicative definitions (where the former is only useful for small domains plaintexts). In this paper we exploit the additive variation. Let $\mathbb{G}$ be a group of prime order p in which DDH is hard. Then the public key is a tuple $\text{PK} = \langle \mathbb{G}, p, g, h \rangle$ and the corresponding secret key is $\text{SK} = s$, s.t. $g^s = h$. Encryption is performed by choosing $r \leftarrow \mathbb{Z}_p$ and computing $\mathsf{Enc}_{\mathsf{PK}}(m; r) = \langle g^r, h^r \cdot g^m \rangle$. Decryption of a ciphertext $c = \langle \alpha, \beta \rangle$ is performed by computing $g^m = \beta \cdot \alpha^{-s}$ and then finding m by running an exhaustive search. Consequently, this variant is only applicable for small plaintext domains, which is the case in our work.

Threshold El Gamal. In El Gamal the parties first agree on a group $\mathbb{G}$ of order p and a generator g. Then, each party P_i picks $s_i \leftarrow \mathbb{Z}_p$ and sends $h_i = g^{s_i}$ to the others. Finally, the parties compute $h = \prod_{i=1}^{n} h_i$ and set $\text{PK} = \langle \mathbb{G}, p, g, h \rangle$. Clearly, the secret key $s = \sum_{i=1}^{n} s_n$ associated with this public key is correctly shared amongst the parties. In order to ensure correct behavior, the parties must prove knowledge of their s_i by running on (g, h_i) the zero-knowledge proof π_{DL}, specified in Section 2.5. To ensure simulation based security, each party must commit to its share first and decommit this commitment only after the commit phase is completed. Note that the simulator can enforce the public key outcome by rewinding the corrupted parties after seeing their decommitment information.

Moreover, decryption of a ciphertext $c = \langle c_1, c_2 \rangle$ follows by computing $c_2 \cdot (\prod_{i=1}^{n} c_1^{s_i})^{-1}$, where each party sends c_1 to the power of its share together with a corresponding proof for proving a Diffie-Hellman relation. Here the simulator can cheat in the proof and return a share of the form $c_2 / (m_{\mathcal{S}} \cdot \left(\prod_{i \in \mathcal{I}} c_1^{s_i}\right)$ where $\mathcal{I}$ is the set of corrupted parties and $m_{\mathcal{S}}$ is the message to be biased. Note that the simulated share may not distribute as the real share (this happens in case $m_{\mathcal{S}}$ is different than the actual plaintext within c). Indistinguishability can be shown by a reduction to the DDH hardness assumption.

The variation $\mathcal{F}_{\text{DecZero}}$ allows the parties to learn whether a ciphertext $c = \langle \alpha, \beta \rangle$ encrypts zero or not, but nothing more. This can be carried out as follows. Each party first raises c to a random non-zero power and rerandomizes the result (proving correctness using a zero-knowledge proof). The parties then decrypt the final ciphertext which is generated by multiplying all the ciphertexts componentwise, and conclude that $m = 0$ if and only if the masked plaintext was g^0.

2.3.4 The Paillier PKE

The Paillier encryption scheme [Pai99] is another example of a public-key encryption scheme that meets Definition 2.7. We focus our attention on the following, widely used, variant of Paillier due to Damgård and Jurik [DJ01]. Specifically, the key generation algorithm chooses two equal length primes p and q and computes $N = pq$. It further picks an element $g \in \mathbb{Z}^*_{N^{s+1}}$ such that $g = (1+N)^j r^N \bmod N^{s+1}$ for a known j relatively prime to N and r^N. Let λ be the least common multiple of $p-1$ and $q-1$, then the algorithm chooses d such that $d \bmod N \in \mathbb{Z}^*_N$ and $d = 0 \bmod \lambda$. The public key is N, g and the secret key is d. Next, encryption of a plaintext $m \in \mathbb{Z}_{N^s}$ is computed by $g^m r^{N^s} \bmod N^{s+1}$. Finally, decryption of a ciphertext c follows by first computing $c^d \bmod N^{s+1}$ which yields $(1+N)^{jmd \bmod N^s}$, and then computing the discrete logarithm of the result relative to $(1+N)$ which is an easy task.

In this work we consider a concrete case where $s = 1$. Thereby, encryption of a plaintext m with randomness $r \leftarrow_R \mathbb{Z}^*_N$ ($\mathbb{Z}_N$ in practice) is computed by,

$$\mathsf{Enc}_N(m, r) = (N+1)^m \cdot r^N \bmod N^2.$$

Finally, decryption is performed by,

$$\mathsf{Dec}_{sk}(c) = \frac{[c^{\phi(N)} \bmod N^2] - 1}{N} \cdot \phi(N)^{-1} \bmod N.$$

The security of Paillier is implied by the Decisional Composite Residuosity (DCR) hardness assumption.

Threshold Paillier. The threshold variant of Paillier PKE in the semi-honest setting can be found in [Gil99], where the parties mutually generate an RSA composite N. A malicious variant realizing this functionality can be found in [HMRT12]. These protocols are fully simulatable in the two-party setting, but can be naturally extended to the multi-party setting (in fact, Hazay et al. also shows a variant that applies for any number of parties). In addition to a key generation protocol, Hazay et al. also designed a threshold decryption protocol which allows to bias the plaintext as required above. The variation π_{DecZero} can be concluded from π_{DEC} as in El Gamal by raising the ciphertext to a random value.

2.3.5 The [BBS04] PKE

To setup the keys we choose at random $x, y \leftarrow \mathbb{Z}^*_p$. The public key is (f, h) where $f = g^x, h = g^y$, and the secret key is (x, y). To encrypt a message $m \in \mathbb{G}$ we choose $r, s \leftarrow \mathbb{Z}_p$ and let the ciphertext be $(u, v, w) = (f^r, h^s, g^{r+s} \cdot m)$. To decrypt a ciphertext $(u, v, w) \in \mathbb{G}^3$ we compute $m = \mathsf{Dec}(u, v, w) = w/u^x v^y$. This homomorphic scheme is IND-CPA secure assuming the hardness of the DLIN assumption and can be viewed as an extension of the El Gamal PKE. Specifically, the protocols we discussed above with respect to El Gamal can be directly extended for this PKE as well.

2.3.6 The [BGN05] PKE

The public key is $\text{PK} = (N, \mathbb{G}, \mathbb{G}_1, e, g, h)$ where $N = q_1 q_2$, $h = u^{q_2}$, g, u are random generators of $\mathbb{G}$, and the secret key is $\text{SK} = q_1$. To encrypt a message $m \in \mathbb{Z}_{q_2}$ we pick a random $r \leftarrow [N-1]$ and compute $g^m h^r$. To decrypt a ciphertext c we observe that $c^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$. Security follows assuming the subgroup decision problem. In a threshold variant, the parties first mutually generate a product of two primes N, so that the factorization of N is shared amongst the parties. To decrypt, each party raises the ciphertext to the power of its share. This scheme supports multiplication in the exponent via the pairing operation, see Definition 2.4. Furthermore, the scheme is additively homomorphic in both groups.

2.4 The Pedersen Commitment Scheme

The Pedersen commitment scheme [Ped91] is defined as follows. A key generation algorithm $(p, g, h, \mathbb{G}) \leftarrow \mathcal{G}(1^\kappa)$ for which the commitment key is $|ck = (\mathbb{G}, p, g, h)$. To commit to a message $m \in \mathbb{Z}_p$ the committer picks randomness $r \leftarrow \mathbb{Z}_p$ and computes $\mathsf{Com}_{\mathsf{CK}}(m; r) = g^m h^r$. The Pedersen commitment scheme is computationally binding under the discrete logarithm assumption, i.e., any two different openings of the same commitment are reduced to computing $\log_g h$. Finally, it is perfectly hiding since a commitment is uniformly distributed in $\mathbb{G}$. Another appealing property of this scheme is its additively homomorphism.

2.5 Zero-Knowledge Proofs

To prevent malicious behavior, the parties must demonstrate that they are well-behaved. To achieve this, our protocols utilize zero-knowledge (ZK) proofs of knowledge. The following proof π_{DL} is required for proving consistency in our maliciously secure threshold decryption protocol. Namely, π_{DL} is employed for demonstrating the knowledge of a solution x to a discrete logarithm problem [Sch89]. Formally stating,

$$\mathcal{R}_{\text{DL}} = \{((\mathbb{G}, g, h), x) \mid h = g^x\}\,.$$

2.6 Hash Functions

The main computational overhead of our basic semi-honest protocol is carried out by P_1, which essentially has to do $m_1 \cdot m_i$ comparisons for each $i \in [2, n]$ in order to compare each of its inputs to each of the other parties' inputs. This overhead can be reduced using hashing, if both parties use the same hash scheme to map their respective items into different $\mathcal{B}$ bins. In that case, the items mapped by some party to a certain bin must only be compared to those mapped by P_1 to the same bin. Thus the number of comparisons can be reduced to be in the order of the number of P_1's inputs times the maximum number of items mapped to a bin. (Of course, care must be taken to ensure that the result of the hashing does not reveal information about the inputs.) In this work we consider two hash schemes: simple hashing and balanced allocations hashing; see [FHNP16] for a thorough discussion.

2.6.1 Simple Hashing

Let h be a randomly chosen hash function mapping elements into bins numbered $1, \ldots, \mathcal{B}$. It is well known that if the hash function h maps m items to random bins, then, if $m \geq \mathcal{B} \log \mathcal{B}$, each bin contains with high probability at most $M = \frac{m}{\mathcal{B}} + \sqrt{\frac{m \log \mathcal{B}}{\mathcal{B}}}$ (see, e.g., [RS98, Wie07]). Setting $\mathcal{B} = m/\log m$ and applying the Chernoff bound shows that $M = O(\log m)$ except with probability $(m)^{-s}$, where s is a constant that depends on the exact value of M.[2]

2.6.2 Balanced Allocation

A different hash construction with better parameters is the balanced allocation scheme of [ABKU99] where elements are inserted into $\mathcal{B}$ bins as follows. Let $h_0, h_1 : \{0,1\}^{p(n)} \to [\mathcal{B}]$ be two randomly chosen hash functions mapping elements from $\{0,1\}^{p(n)}$ into bins $1, \ldots, \mathcal{B}$. An element $x \in \{0,1\}^{p(n)}$ is inserted into

[2] As stated in [FHNP16], by setting $\mathcal{B} = m \log\log m / \log m$ we can make the error probability negligible in m. However, any actual implementation will have to examine the exact value of $\mathcal{B}$ which results in a sufficiently small error probability for the input sizes that are expected. As for theoretical analysis, the subsequent construction, based on balanced allocation hashing, presents a negligible error probability.

the less occupied bin from $\{h_0(x), h_1(x)\}$, where ties are broken arbitrarily. If m elements are inserted, then except with negligible probability over the choice of the hash functions h_0, h_1, the maximum number of elements allocated to any single bin is at most $M = O(m/\mathcal{B} + \log\log\mathcal{B})$. Setting $\mathcal{B} = \frac{m}{\log\log m}$ implies that $M = O(\log\log m)$.[3]

3 The Semi-Honest Construction

We begin with a description of a private MPC protocol that securely realizes the following functionality in the presence of semi-honest adversaries. Specifically, the private set-intersection functionality $\mathcal{F}_{\text{PSI}}$ for n parties is defined by $(X_1, \ldots, X_n) \mapsto (X_1 \cap \ldots, \cap X_n, -, \ldots, -)$. For simplicity we consider a functionality where only the first party receives an output, extending it to the general case is straightforward. Our protocol takes a new approach where party P_1 interacts with every party using a 2PC protocol that implements $\mathcal{F}_{\text{PSI}}$ for two parties. At the end, P_1 combines the results of all these protocols and learns the intersection.

To be concrete, assume that P_1 learns for each element $x_1^j \in X_1$ whether it is in X_i or not, for all $j \in [m_1]$ and $i \in [2, n]$. Then, P_1 can conclude the overall intersection. This is because an element from X_1 that intersects with all other sets must be in the overall intersection. On the other hand, any element that is joint for all sets must be in X_1 as well. Thus, we conclude that it is sufficient to individually compare X_1 with all other sets. This protocol, of course, is insecure as it leaks the pairwise intersections (which is much more information than P_1 should learn from a secure realization of $\mathcal{F}_{\text{PSI}}$). In order to hide this leakage we suggest to use a subprotocol for which P_1 learns an encryption of zero in case the corresponding element is in the intersection, and an encryption of a random element otherwise. If the encryption is additively homomorphic then P_1 can combine all the results with respect to each element $x_1^j \in X_1$, so that x_1^j is in the overall intersection if and only if the combined ciphertext encrypts the zero string. We implement this subprotocol using a variant of the [FNP04] protocol; see below for a complete description.

The [FNP04] protocol (the semi-honest variant). More concretely, the [FNP04] protocol is based on oblivious polynomial evaluation. The basic two-round semi-honest protocol, executed between parties $\widetilde{P}_1$ and $\widetilde{P}_2$ on the respective inputs X_1 and X_2 of sizes m_1 and m_2, works as follows:

1. Party $\widetilde{P}_2$ chooses encryption/decryption keys $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$ for an additively homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

   $\widetilde{P}_2$ further computes the coefficients of a polynomial $Q(\cdot)$ of degree m_2, with roots set to the m_2 elements of X_2, and sends the encrypted coefficients, as well as PK, to $\widetilde{P}_1$.

2. For each element $x_1^j \in X_1$ (in random order), party $\widetilde{P}_1$ chooses a random value r_j (taken from an appropriate set depending on the encryption scheme), and uses the homomorphic properties of the encryption scheme to compute an encryption of $r_j \cdot Q(x_1^j) + x_1^j$. $\widetilde{P}_1$ sends the encrypted values to $\widetilde{P}_2$.

3. Upon receiving these ciphertexts, $\widetilde{P}_2$ extracts $X_1 \cap X_2$ by decrypting each value and then checking if the result is in X_2. Note that if $z \in X_1 \cap X_2$ then by the construction of the polynomial $Q(\cdot)$ we get that $r \cdot Q(z) + z = r \cdot 0 + z = z$ for any r. Otherwise, $r \cdot Q(z) + z$ is a random value that reveals no information about z and (with high probability) is not in X_2.

[3] A constant factor improvement is achieved using the *Always Go Left* scheme in [Vöc03] where $h_0 : \{0,1\}^{p(n)} \to [1, \ldots, \frac{\mathcal{B}}{2}], h_1 : \{0,1\}^{p(n)} \to [\frac{b}{2} + 1, \ldots, \mathcal{B}]$. An element x is inserted into the less occupied bin from $\{h_0(x), h_1(x)\}$; in case of a tie x is inserted into $h_0(x)$.

Towards realizing $\mathcal{F}_{\text{PSI}}$ we slightly modify the [FNP04] protocol as follows. The role of $\widetilde{P}_2$ remains almost the same and played by all parties P_i for $i \in [2, n]$, except that these parties do not generate a pair of keys but rather use a public key that was previously generated by the whole set of parties in a key generation phase. Whereas for each element $x_1^j \in X_1$ (picked in random order), $\widetilde{P}_1$ computes the encryption of $r_j \cdot Q(x_1^j)$ and keeps it for itself. This role is computed by party P_1 that aggregates the polynomial evaluations and concludes the intersection as explained in the beginning of this section. We denote $\widetilde{P}_\tau$'s message sent within this modified protocol by π_{FNP}^τ for $\tau \in \{1, 2\}$.

Our complete protocol. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote a threshold additively homomorphic cryptosystem with a public key generation and decryption protocols $\pi_{\text{GEN}}^{\text{SH}}$ and $\pi_{\text{DEC}}^{\text{SH}}$, respectively (in fact, we will be using protocol $\pi_{\text{DecZero}}^{\text{SH}}$; see Section 2.3.3). Then our protocol can be described using three phases. In the first phase the parties run protocol $\pi_{\text{GEN}}^{\text{SH}}$ in order to agree on a public key without disclosing its corresponding secret key to anyone. In the second 2PC phase P_1 individually interacts with each party in order to generate the set of ciphertexts as specified above (via the [FNP04] modified protocol). Finally, in the last phase, the parties carry out protocol $\pi_{\text{DecZero}}^{\text{SH}}$ for which P_1 concludes the overall intersection. More formally,

Protocol 1 (Protocol π_{PSI} with semi-honest security)**.**

- **Input:** *Party P_i is given a set $X_i = \{x_i^1, \ldots, x_i^{m_i}\}$ of size m_i for all $i \in [n]$. All parties are given a security parameter 1^κ and a description of a group $\mathbb{G}$.*
- **The protocol:**
  - **Key Generation.** *The parties mutually generate a public key* PK *and the corresponding secret key shares $(\text{SK}_1, \ldots, \text{SK}_n)$ by running a semi-honestly secure protocol $\pi_{\text{GEN}}^{\text{SH}}$ that realizes $\mathcal{F}_{\text{GEN}}$.*
  - **The 2PC phase.** *Party P_1 engages in an execution of protocol $(\pi_{\text{FNP}}^1, \pi_{\text{FNP}}^2)$ specified above with each party P_i, for every $i \in [2, n]$. Let $(c_1^i, \ldots, c_{m_i}^i)$ denote the set of ciphertexts sent by party P_i.*
  - **Concluding the intersection.**
    1. *Upon receiving the ciphertexts from all parties, party P_1 combines the following ciphertexts*

$$c_1 = \prod_{i=2}^{n} c_1^i, \ldots, c_{m_{\text{MAX}}} = \prod_{i=2}^{n} c_{m_{\text{MAX}}}^i$$

       where $m_{\text{MAX}} = \max(m_2, \ldots, m_n)$. Note that P_1 calculates the ciphertexts encrypting the coefficients of the combined polynomial $Q_1 = Q_2(\cdot) + \cdots + Q_n(\cdot)$.
    2. *The parties mutually decrypt for P_1 the set of ciphertexts $c_1^1 = r_1 \cdot Q_1(x_1^1), \ldots, c_1^{m_1} = r_{m_1} \cdot Q_1(x_1^{m_1})$ by engaging in a semi-honestly secure protocol $\pi_{\text{DecZero}}^{\text{SH}}$ that realizes $\mathcal{F}_{\text{DecZero}}$.*
    3. *P_1 outputs x_1^j only if the decryption of c_1^j equals zero.*

We continue with the proof of the following theorem,

Theorem 3.1. *Assume that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure threshold additively homomorphic encryption scheme. Then, Protocol 1 securely realizes $\mathcal{F}_{\text{PSI}}$ in the presence of semi-honest adversaries in the $\{\mathcal{F}_{\text{GEN}}, \mathcal{F}_{\text{DecZero}}\}$-hybrid for $n \geq 2$ parties.*

Proof: We already argued for correctness, we thus directly continue with the privacy proof. We consider two classes of adversaries. The first class involves adversaries that corrupt a subset of parties that includes party P_1, whereas the second class does not involve the corruption of P_1. We provide a separate simulation for each class.

Consider an adversary $\mathcal{A}$ that corrupts a strict subset $\mathcal{I}$ of parties from the set $\{P_1, \ldots, P_n\}$, including P_1. We define a simulator $\mathcal{S}$ as follows.

1. Given $\{X_i\}_{i\in\mathcal{I}}$ and $Z = \cap_{i=1}^{n} X_i$, the simulator invokes the corrupted parties on their corresponding inputs and randomness.

2. $\mathcal{S}$ generates $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$ and invokes the simulator $\mathcal{S}_{\mathrm{GEN}}(\mathrm{PK})$ for $\pi^{\mathrm{SH}}_{\mathrm{GEN}}$ in the key generation phase.

3. Next, $\mathcal{S}$ plays the role of the honest parties against P_1 on arbitrary sets of inputs. Namely, $\mathcal{S}$ sends ciphertexts encrypting the polynomials induced by these inputs.

4. Finally, at the concluding phase the simulator completes the decryption protocol as follows. For each $x_1^j \in Z$, $\mathcal{S}$ invokes $\mathcal{S}_{\mathrm{DecZero}}(0)$, forcing the decryption outcome to be zero. Whereas for each $x_1^j \notin Z$, the simulator invokes $\mathcal{S}_{\mathrm{DecZero}}(r)$ for a uniformly distributed $r \leftarrow \mathbb{G}$.

Note that the difference between the two views is with respect to the encrypted polynomials sent by the simulator as opposed to the real parties. Then indistinguishability follows from the privacy of π_{DecZero} which boils down to the privacy of the threshold homomorphic encryption scheme. This can be shown via a reduction to the indistinguishability of ciphertexts of the encryption scheme. More formally, assume by construction the existence of an adversary $\mathcal{A}$ and a distinguisher D that distinguishes the real and simulated executions with non-negligible probability. We construct an adversary $\mathcal{A}_\Pi$ that distinguishes two sets of ciphertexts. Concretely, upon receiving a public key PK, $\mathcal{A}_\Pi$ invokes the simulator $\mathcal{S}_{\mathrm{GEN}}(\mathrm{PK})$ as would the simulator $\mathcal{S}$ do. Next, it outputs two sets of vectors. One corresponds to the set of polynomials computed from the honest parties' inputs. Whereas the other set is arbitrarily fixed as generated in the simulation. Upon receiving the vector of ciphertexts $\tilde{c}$ from its oracle, $\mathcal{A}_\Pi$ sends $\tilde{c}$ to the corrupted P_1 and completes the reduction as in the simulation.

Note that if $\tilde{c}$ corresponds to encryptions of the honest parties' inputs, then the adversary's view is distributed as in the real execution. In particular, $\mathcal{A}_\Pi$ always knows the correct plaintext to be decrypted (which is either zero or a random value where this randomness is also known in the semi-honest model). Therefore, the shares handed by $\mathcal{A}_\Pi$ are as in the real execution. On the other hand, in case $\tilde{c}$ corresponds to the set of arbitrary inputs, then the adversary's view is distributed as in the simulation since the decrypted plaintext is not correlated with the actual plaintext. This concludes the proof.

Next, we consider an adversary which does not corrupt P_1. In this case the simulator $\mathcal{S}$ is defined as follows.

1. Given $\{X_i\}_{i\in\mathcal{I}}$ and $Z = \cap_{i=1}^{n} X_i$, the simulator invokes the corrupted parties on their corresponding inputs and randomness.

2. $\mathcal{S}$ generates $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$ and invokes the simulator $\mathcal{S}_{\mathrm{GEN}}(\mathrm{PK})$ for π_{GEN} in the key generation phase.

3. Next, $\mathcal{S}$ plays the role of P_1 against the corrupted parties on an arbitrary set of inputs and concludes the simulation by playing the role of P_1 on these arbitrary inputs. (Note that this corruption case is even simpler as only P_1 learns the output. In case all parties should learn the output then we apply the same simulation technique as in the previous corruption case.)

Note that the difference is with respect to the polynomial evaluations made by the simulated P_1 which uses an arbitrary input. Then the indistinguishability argument follows similarly as above via a reduction to the privacy of the encryption scheme as only P_1 receives an output. ■

3.1 Communication and Computation Complexities

Note that the complexity of the protocol is dominated by the overhead of the threshold cryptosystem as well as the underlying 2PC protocol for implementing $\mathcal{F}_{\text{PSI}}^{\text{2PC}}$. We instantiate the latter using the [FNP04] and either the El Gamal PKE [Gam85] or the Paillier PKE [Pai99] for the former. Note that the communication complexity of the [FNP04] variant we consider here is linear in m_2, as $m_2 + 1$ encrypted values are sent from $\widetilde{P}_2$ to $\widetilde{P}_1$ (these are the encrypted coefficients of $Q(\cdot)$). However, the work performed by $\widetilde{P}_1$ is high, as each of the m_1 oblivious polynomial evaluations includes performing $O(m_2)$ exponentiations, totaling in $O(m_1 \cdot m_2)$ exponentiations. To save on computational work, Freedman et al. introduced hash functions into their schemes. Below we consider two instantiations of simple hashing (cf. Section 2.6.1) and balanced allocation hash function (cf. Section 2.6.2).

Furthermore, the underlying threshold additively homomorphic encryption scheme can be instantiated using either the additive variant of the El Gamal PKE, for which the public key can be generated using the Diffie-Hellman approach [DH76], or the Paillier PKE for which the public key can be generated using [Gil99]. Finally, we note that our protocol is constant round and does not need to use any broadcast channel.

Improved computation using simple hashing. In our protocol, the hash function h will be picked by one of the parties (say $\widetilde{P}_2$) and known to both. Moreover, $\widetilde{P}_2$ defines a polynomial of degree M for each bin by fixing its mapped elements to be the set of roots. As some of the bins contain less than M elements, $\widetilde{P}_2$ pads each polynomial with zero coefficients up to degree M, so that the total degree of the polynomial is M (since P_2 must hide the actual number of elements allocated to each bin). This results in $\mathcal{B}$ polynomials, all of degree M, with exactly m_2 non-zero roots. The rest of the protocol remains unchanged. Now, $\widetilde{P}_1$ needs to first map each element x_1^j in its set to its bin and then obliviously evaluate the polynomial that corresponds to that bin. Neglecting small constant factors, the communication complexity is not affected as $\widetilde{P}_2$ (emulated by party P_i for $i \in [2, n]$) now sends $\mathcal{B} \cdot M_i = O(m_i)$ encrypted values. There is, however, a dramatic reduction in the work performed by $\widetilde{P}_1$ as each of the oblivious polynomial evaluations amounts now to performing just $O(M_i)$ exponentiations, and hence $\widetilde{P}_1$ performs $O(m_1 \cdot \sum_i M_i)$ exponentiations overall, where M_i is the bin size for allocating P_1's input in the execution with P_i.

Improved computation using balanced allocation hashing. Loosely speaking, they used the balanced allocation scheme of [ABKU99] with $\mathcal{B} = \frac{m_2}{\log\log m_2}$ bins, each of size $M = O(m_2/\mathcal{B} + \log\log\mathcal{B}) = O(\log\log m_2)$. Party $\widetilde{P}_2$ now uses the balanced allocation scheme to hash every $x \in X$ into one of the $\mathcal{B}$ bins resulting (with high probability) with each bin's load being at most M. Instead of a single polynomial of degree m_2 party $\widetilde{P}_2$ now constructs a degree-M polynomial for each of the $\mathcal{B}$ bins, i.e., polynomials $Q_1(\cdot), \ldots, Q_{\mathcal{B}}(\cdot)$ such that the roots of $Q_i(\cdot)$ are the elements put in the i^{th} bin. Upon receiving the encrypted polynomials, party $\widetilde{P}_1$ obliviously evaluates the encryption of $r_0^j \cdot Q_{h_0(x_j^1)}(x_j^1)$ and $r_1^j \cdot Q_{h_1(x_j^1)}(x_j^1)$ for each of the two bins $h_0(x_j^1), h_1(x_j^1)$ in which x_j^1 can be allocated, enabling $\widetilde{P}_1$ to extract $X \cap Y$ as above.

The communication and computational overheads are as above. Nevertheless, a subtlety emerges in our semi-honest protocol that employs this tool, as P_1 cannot tell which of the two bins contains the particular element. Consequently, it cannot tell which of the two associated polynomials is evaluated to zero, where this information is crucial in order to conclude the intersection. We suggest two solutions in order to overcome this issue. Our first solution supports the El Gamal and Paillier PKEs but requires more communication. Namely, the parties run a protocol to compute the encryption of the product of plaintexts. This is easily done by having $\widetilde{P}_1$ additively mask the two evaluations and then have $\widetilde{P}_2$ multiply the decrypted results

and send the encrypted product back to $\widetilde{P}_1$. At the end, $\widetilde{P}_1$ unmasks this cipehrtext and continues with the protocol execution. Note that all the products can be computed in parallel.

Our second solution uses an encryption scheme that is additively homomorphic and multiplicative with respect to a single plaintexts multiplication. In this case, it is possible to multiply the two results of the polynomials evaluations, which will result zero if one of the evaluations is zero. An additively homomorphic encryption scheme that supports such a property is due to Boneh et al. [BGN05] (cf. Section 2.3.6), or due to Catalano and Fiore [CF15] which only requires a mild property of the underlying additively homomorphic scheme yet the number of additions of degree-2 terms is bounded by a constant (which corresponds to the number of parties in our scheme).

4 The Malicious Construction

Towards designing a protocol with stronger security we need to handle new challenges that emerge due to the fact that party P_1 may behave maliciously. The main challenge is to prevent P_1 from learning additional information about the intersection with individual parties. To be concrete, we recall that our semi-honest protocol follows by having P_1 individually interacting with each party via 2PC protocol, where this stage is followed by decrypting the combined ciphertexts generated in these executions. Then upon corrupting a subset of parties which includes P_1, a malicious adversary may use ill formed ciphertexts or ciphertexts for which it does not know their corresponding plaintext, exploiting the honest parties as a decryption oracle. Towards dealing with malicious attacks we modify Protocol 1 as follows (for simplicity we concretely consider the El Gamal PKE and adapt our ZK proofs for this encryption scheme).

1. First, P_1 broadcasts commitments to its input X_1 together with a zero-knowledge proof. This phase is required in order to ensure that P_1 uses the *same input* against every underlying 2PC evaluation with every other party. One particular instantiation for this commitment scheme can be based in Pedersen's scheme (cf. Section 2.4). This scheme is consistent with El Gamal PKE (cf. 2.3.3) and the BBS PKE (cf. 2.3.5). An alternative scheme, e.g. [DN02], can be considered when using the Paillier or the BGN PKEs (cf. Sections 2.3.4 and 2.3.6, respectively); see below for more details.

2. To prevent P_1 from cheating when assembling the encrypted polynomial, each party chooses a random element $\lambda_i \leftarrow \mathbb{G}$ and encrypts the product of each coefficient of $Q_i(\cdot)$ with λ_i. More specifically, P_i sends an encryption of polynomial $\lambda_i \cdot Q_i(\cdot)$, where the underlying set of roots remains unchanged. This later allows the other parties to verify the correctness of P_1's computation, which will allow to claim that P_1 can only learn a random group element upon deviating.

3. Next, the parties pick a random group element $u \leftarrow \mathbb{G}$ and compare the evaluation of P_1's combined polynomial against the evaluations of their own individual polynomials. Namely, each party broadcasts the value $\tilde{\lambda}_i = \prod_j (c_j^i)^{u^j}$ together with a zero-knowledge proof of knowledge. If concluded correctly, this phase is followed by the parties verifying the equality of the following equation

$$\prod_{j=1}^{m_{\text{MAX}}} (c_j)^{u^j} = \sum_{i=2}^{n} \tilde{\lambda}_i$$

where m_{MAX} is the maximum over all input sets sizes, n is the number of parties and $\left(c_1, \ldots, c_{m_{\text{MAX}}}\right)$ are the ciphertexts combined by P_1. Note that the equality check is performed over the ciphertexts. For this reason we can only work with additively homomorphic PKEs for which the homomorphic

operation does not add noise to the ciphertext. Our crucial observation here is that the simulator can run the extractor of the proof of knowledge and obtain the polynomials evaluations. Now, if the adversary convinces the honest parties with a non-negligible probability that it indeed knows the plaintext, then the simulator can rewind it sufficiently many times in order to extract enough evaluation points for which it can fully recover the corrupted parties' polynomials, and hence their inputs.

4. Finally, P_1 must prove that it correctly evaluated the combined polynomial on its committed input X_1 from Item 1. This phase is backed up with a ZK proof due to Bayer and Groth [BG13], denoted by π_{EVAL}, and formally stated in Section 2.5.

Sub-protocols. Our protocol uses the following sub-protocols.

1. A coin tossing protocol π_{COIN} employed in order to sample a random group element $u \leftarrow \mathbb{G}$. Our protocol employs π_{COIN} only once, where u is locally substituted by the parties in their private polynomials. These values are then used by the parties to verify the behaviour of P_1. The overhead of π_{COIN} is $O(n^2)$ where n is the number of parties. Looking ahead, we instantiate our primitives in this work in either the Random Oracle (RO) model or in the Common Reference String (CRS) model. In the former (programmable) RO model, a simple UC commitment to a message m instantiation is computed by fixing $\text{RO}(r\|m)$ for some nonce r. Whereas in the CRS model, we can implement efficient UC commitments using $O(1)$ group elements for the global CRS and $O(1)$ group elements for each commitment [Lin11]. Next, we can use UC commitments within the standard Blum's protocol. We stress that our construction does not achieve UC security due to the use of rewinding in our proof.

2. A ZK proof of knowledge π_{EXP} for demonstrating the knowledge of the message with respect to an additively homomorphic commitment scheme. We employ this proof in two distinct places in our protocol, and for two different purposes. First, when P_1 broadcasts its polynomial in Step 2 and proves the knowledge of these coefficients and second, in Step 4c when each party sends its polynomial evaluation. As we demonstrate below, for both instantiations we can use the same proof for the two purposes. Importantly, since we are in the multi-party setting, where each party uses a homomorphic encryption to encrypt its polynomial, we must avoid the case for which an adversary may "reuse" one of the encrypted polynomials as the polynomial of one of the corrupted parties. We will require the proof to be many-many simulation-extractable. We will achieve this by showing that our proofs are simultaneously straight-line simulatable and extractable. One of our instantiations will be UC secure and the other instantiation will be based on random oracles.

3. A ZK proof of knowledge π_{EVAL} for demonstrating the correctness of a polynomial evaluation for a secret committed value. We will require this proof to be a stand-alone zero-knowledge argument of knowledge. We will instantiate it using [BG13]. This proof is an argument of knowledge such that given a polynomial $P(\cdot) = (p_0, \ldots, p_d)$ and two commitments $\mathsf{com}, \mathsf{com}'$, proves the knowledge of a pair v, u such that $P(v) = u$ where $\mathsf{com} = \mathsf{Com}(u)$, $\mathsf{com}' = \mathsf{Com}(v)$ and $\mathsf{Com}(\cdot)$ denotes an homomorphic commitment scheme (as noted in [BG13] any homomorphic commitment can be used). Moreover, the polynomial can be committed as well. Formally stating,

$$\mathcal{R}_{\text{EVAL}} = \left\{ \left(P(\cdot) = (p_0, \ldots, p_d), \mathsf{com}, \mathsf{com}'\right), (r, r', u, v) \mid \begin{array}{r} \mathsf{com} = \mathsf{Com}(u; r) \\ \wedge\, \mathsf{com}' = \mathsf{Com}(v; r') \\ \wedge\, P(u) = v \end{array} \right\}.$$

Importantly, the communication complexity of this proof is logarithmic in the degree of the polynomial, whereas the computational overhead by the verifier is $O(d)$ multiplications.

We next formally describe our protocol.

Protocol 2 (Protocol π_{ML} with malicious security)**.**

- **Input:** *Party P_i is given a set $X_i = \{x_i^1, \ldots, x_i^{m_i}\}$ of size m_i for all $i \in [n]$. All parties are given a security parameter 1^κ and a description of a group $\mathbb{G}$.*
- **The protocol:**
  1. **Key Generation.** *The parties mutually generate a public key* PK *and the corresponding secret key shares* $(\text{SK}_1, \ldots, \text{SK}_n)$ *by running a maliciously secure protocol $\pi_{\text{GEN}}^{\text{ML}}$ that realizes $\mathcal{F}_{\text{GEN}}$.*
  2. **The commitment phase.** *P_1 creates commitments to its inputs $\{\mathsf{com}_1, \ldots, \mathsf{com}_{m_1}\}$ and broadcasts them to all parties and proves the knowledge of their decommitments using threshold π_{EXP}.*
  3. **The 2PC phase.** *For all $i \in [2, n]$, party P_i computes the coefficients of a polynomial $Q_i(\cdot) = (q_0^i, \ldots, q_{m_i}^i)$ of degree m_i, with roots set to the m_i elements of X_i. In addition, P_i chooses a random element $\lambda_i \leftarrow \mathbb{G}$ and computes the product $\lambda_i \cdot q_j^i$ for every coefficient within Q_i. Finally, P_i sends P_1 the sets of ciphertexts $\left(c_1^i, \ldots, c_{m_i}^i\right)$, encrypting the coefficients of $\lambda_i \cdot Q_i(\cdot)$.*
  4. **Concluding the intersection.**
     (a) *Upon receiving the ciphertexts from all parties, party P_1 combines the following ciphertexts*

$$c_1 = \prod_{i=2}^{n} c_1^i, \ldots, c_{m_{\text{MAX}}} = \prod_{i=2}^{n} c_{m_{\text{MAX}}}^i$$

where $m_{\text{MAX}} = \max(m_2, \ldots, m_n)$. Note that P_1 calculates the ciphertexts encrypting the coefficients of the combined polynomial $\lambda_2 \cdot Q_2(\cdot) + \cdots + \lambda_n \cdot Q_n(\cdot)$. P_1 then broadcasts ciphertexts $\left(c_1, \ldots, c_{m_{\text{MAX}}}\right)$ to all parties.

     (b) *Next, the parties verify the correctness of these ciphertexts. Specifically, the parties first agree on a random element u from the appropriate plaintext domain using the coin tossing protocol π_{COIN}.*

     (c) *Then, each party broadcasts the ciphertext computed by $\prod_j (c_j^i)^{u^j}$, denoted by $\tilde{\lambda}_i$, together with a ZK proof of knowledge π_{EXP} for proving the knowledge of the plaintext.*
     If all the proofs are verified correctly, then the parties check that $\prod_{j=1}^{m_{\text{MAX}}} (c_j)^{u^j} = \sum_{i=2}^{n} \tilde{\lambda}_i$ using the homomorphic property of the encryption scheme.

     (d) *If the verification phase is completed correctly, for every $x_1^j \in X_1$, P_1 evaluates the polynomial that is induced by the coefficients encrypted within ciphertexts $\left(c_1, \ldots, c_{m_{\text{MAX}}}\right)$ on x_1^j and proves consistency with the commitments from Step 2 using the ZK proof π_{EVAL}.*

     (e) *Upon completing the evaluation, the parties decrypt the evaluation outcomes for P_1 using protocol $\pi_{\text{DecZero}}^{\text{ML}}$, who concludes the intersection.*

We continue with the proof for this theorem,

Theorem 4.1. *Assume that* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CPA secure threshold additively homomorphic encryption scheme, and that $\pi_{\text{COIN}}, \pi_{\text{EXP}}, \pi_{\text{EVAL}}, \pi_{\text{GEN}}$ and π_{DecZero} are as above. Then, Protocol 2 securely realizes $\mathcal{F}_{\text{PSI}}$ in the presence of malicious adversaries for $n \geq 2$ parties.*

Proof: Intuitively, correctness follows easily due to a similar argument as in the semi-honest case , where each element in P_1's set must zero all the other polynomials if it belongs to the intersection. Next, we consider two cases depending on the set of parties corrupted by the adversary.

- Case 1: The set of parties $\mathcal{I}$ corrupted by the adversary includes P_1.
- Case 2: The set of parties $\mathcal{I}$ corrupted by the adversary does not include P_1.

Next, we consider each case and prove correctness.

Case 1: Corrupted set includes P_1. Consider an adversary $\mathcal{A}$ that corrupts a strict subset $\mathcal{I}$ of parties from the set $\{P_1, \ldots, P_n\}$, including P_1. We define a simulator $\mathcal{S}$ as follows.

1. Given $\{X_i\}_{i \in \mathcal{I}}$ the simulator invokes the corrupted parties on their corresponding inputs and randomness.
2. $\mathcal{S}$ generates $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$ and invokes the simulator $\mathcal{S}_{\mathrm{GEN}}(\mathsf{PK})$ for $\pi^{\mathrm{ML}}_{\mathrm{GEN}}$ in the key generation phase.
3. Next, $\mathcal{S}$ extracts the input X'_1 of P_1 by invoking the extractor of the proof of knowledge π_{EXP}.
4. $\mathcal{S}$ plays the role of the honest parties against P_1 on arbitrary sets of inputs.
5. $\mathcal{S}$ completes the protocol execution until Step 4d and aborts, sending $\perp$ to the trusted party if one of the proofs fails.
6. Otherwise, $\mathcal{S}$ extracts the corrupted parties' inputs (excluding party P_1 for which its input has already been extracted). More concretely, the simulator repetitively rewinds the adversary to the beginning of Step 4b, where for every iteration the parties evaluate their polynomial at a randomly chosen point u and the simulator extracts the individual evaluations by running the extractor of the proof of knowledge π_{EXP} and records these values only if they pass the verification phase.

   Upon recording $d+1$ values for each corrupted party, the simulator reconstructs the combined polynomials of all corrupted parties. It adds the values extracted for each party $i \in \mathcal{I}$ and uses the combined values for each of the $d+1$ points u to reconstruct a polynomial using polynomial interpolation. Then the simulator obtains the roots of the polynomials, say X', which it sets as the combined input of the corrupted parties. In other words, the roots of this polynomial can be thought of as the intersection of elements of the corrupted parties excluding party P_1's input that has already been extracted. S sends the combined input X' as the input of each corrupted party (excluding P_1's). For P_1, it sends the input extracted in the first step.
7. $\mathcal{S}$ receives the set intersection Z from the trusted party.
8. Finally, for every $x^j_1 \in Z$, $\mathcal{S}$ enforces the decryption to be zero, whereas for $x^j_1 \notin Z$ the simulator sets the decryption as a random group element. It then runs the simulator $\mathcal{S}^{\mathrm{ML}}_{\mathrm{DecZero}}$ on the appropriate plaintext, PK and the corrupted parties' shares of the secret key.
9. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

We briefly discuss the running time of the simulator. Observe that its running time is dominated by Step 6, when it repeatedly rewinds the adversary. Nevertheless, using a standard analysis, the expected number of rewindings can be shown to be polynomial. We next prove that the real and simulated executions are computationally indistinguishable. Note that the difference between the executions boils down to the privacy of the encryption scheme. Namely, the simulator sends encryptions of polynomials that were computed based on arbitrary inputs, as opposed to the honest parties' real inputs. Our proof follows via a sequence of hybrid games. We will begin with a scenario where P_1 is in the set of corrupted parties $\mathcal{I}$. When P_1 is honest, the proof is simpler and we discuss this at the end.

$\mathbf{Hybrid}_0$: The first game is the real execution.

$\mathbf{Hybrid}_1$: This hybrid is identical to the real world with the exception that the simulator $\mathcal{S}_1$ in this experiment extracts the corrupted parties inputs as in the simulation. More precisely, it extracts the inputs of all corrupted parties from π_{EXP}, and aborts if it fails to extract. Then, it computes X and X' the inputs of P_1 and the combined inputs of the rest of corrupted parties. It checks if the final output is correct w.r.t. X being P_1's input and X' being the inputs of the other corrupted parties. It can check this, because in this hybrid, the simulator knows the actual inputs of the honest parties. If the output is incorrect, the simulator aborts.

To argue indistinguishability of $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$, it suffices to show that the simulator aborts only with negligible probability.

As proofs made using π_{EXP} are straight-line extractable, we have that, except with negligible probability, the simulator will obtain the messages in the commitments. If the corrupted parties complete the proof only with negligible probability, then it follow directly that $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ are statistically close. Otherwise, we have that all corrupted parties complete their proofs with non-negligible probability. Since the evaluation points u come from a field of size super-polynomial, we have that the simulator will be able to obtain $d+1$ points on which all corrupted parties give convincing proofs with non-negligible probability. This in turn means that, except with negligible probability, the simulator can extract the plaintexts corresponding $\prod_j (c_j^i)^{u^j}$. Therefore, via interpolation it can reconstruct the coefficients of the combined polynomial. Furthermore, it can obtain X' by obtaining the combined polynomial of only the corrupted parties (excluding X') (following the strategy of the real simulator) by subtracting the polynomials of the honest parties. Finally, using the soundness of the π_{EVAL} and $\pi^{\text{ML}}_{\text{DecZero}}$ and the binding property of the commitments made in the first step, we have that extracted values will be consistent with the computation performed in the last step. Therefore, the simulator aborts only with negligible probability.

$\mathbf{Hybrid}_2$: In this hybrid, the simulator extracts just as in $\mathbf{Hybrid}_1$ with the following modifications. First, it invokes simulator $\mathcal{S}_{\text{GEN}}$ for protocol π_{GEN} in Step 1. In addition, if the simulator does not abort when executing Step 4b, it computes the set-intersection result Z based on the extracted inputs and the honest parties' inputs (which it knows in this hybrid). Next, it invokes simulator $\mathcal{S}_{\text{DecZero}}$ of the decryption protocols that is invoked in Step 4e. Note that $\mathcal{S}_{\text{DecZero}}$ is handed as plaintexts result of the set-intersection and needs to bias the outcome towards these set of plaintexts. That is, for each element $z \in X_1$ substituted in the combined polynomial in Step 4d, the simulator enforces the decryption to be zero, and a random element otherwise. Note that indistinguishability follows from the properties of the threshold decryption. In particular, the adversary's view in the previous hybrid includes the real execution of protocols π_{GEN} and π_{DEC}, whereas in the current hybrid the adversary's view includes the simulated protocols executions. We proved correctness in the previous hybrid and correctness of this hybrid follows from indistinguishability of the two hybrids.

$\mathbf{Hybrid}_3$: In this hybrid, the simulator changes all the proofs given by the honest parties in Step 4b to simulated ones. Moreover, recall that the simulator continues to extract the inputs of the corrupted parties. Now, since the zero-knowledge proof we employ in this step is straight-line simulatable, it follows that $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ are computationally indistinguishable.

$\mathbf{Hybrid}_4$: In this hybrid, the simulator changes the inputs of the honest parties in the 2PC phase to random inputs. Namely, the simulator sends the encryptions of a random polynomial on behalf of each honest party in Step 3. Then indistinguishability of $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ follows from the IND-CPA security of the underlying encryption scheme. Specifically, the simulator never needs to know the secret key of the encryption scheme, so that the ciphertexts obtained from the encryption oracle in the IND-CPA reduction can be directly plugged into the protocol. More concretely, a simple reduction can follow by providing an adversary $\mathcal{A}'$, who wishes to break the IND-CPA security of the underlying PKE, a public-key PK

and a sequence of ciphertexts that either encrypt the real honest parties' polynomials or a set of random polynomials. $\mathcal{A}'$ emulates the simulator for this hybrid, with the exception that it plugs-in these ciphertexts on behalf of the honest parties in Step 3. Note that the adversary's view is either distributed according to $\mathbf{Hybrid}_4$ or $\mathbf{Hybrid}_3$, where no information about the polynomials is revealed in Step 4c as each polynomial is masked with a random element λ_i and thus the evaluation in a random point u will be close to the uniform distribution.

As $\mathbf{Hybrid}_4$ is identical to the real simulator, the proof of indistinguishability follows via a standard hybrid argument. This concludes case 1 and we described case 2 next.

Case 2: Corrupted set does not include P_1. Next, in the case that P_1 is not corrupted, the simulator further plays the role of this party in the simulation. In this case the proof follows almost as above with the difference that now the simulator uses a fake input for P_1 when emulating Step 4d and will fake the proofs made using π_{EXP} and π_{EVAL} from P_1.

More precisely, consider an adversary $\mathcal{A}$ that corrupts a strict subset $\mathcal{I}$ of parties from the set $\{P_1, \ldots, P_n\}$, excluding P_1. We define a simulator $\mathcal{S}$ as follows.

1. Given $\{X_i\}_{i\in\mathcal{I}}$ the simulator invokes the corrupted parties on their corresponding inputs and randomness.

2. $\mathcal{S}$ generates $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^\kappa)$ and invokes the simulator $\mathcal{S}_{\text{GEN}}(\text{PK})$ for $\pi^{\text{ML}}_{\text{GEN}}$ in the key generation phase.

3. Next, $\mathcal{S}$ commits to all 0 for P_1's commitments and fakes the proof of knowledge π_{EXP}.

4. $\mathcal{S}$ plays the role of the honest parties against P_1 on arbitrary sets of inputs.

5. Finally, at the concluding phase the simulator completes the execution of the protocol as follows. $\mathcal{S}$ completes the protocole execution until Step 4d. If and of the proofs fail $\mathcal{S}$ aborts, sending $\perp$ to the trusted party.

6. Otherwise, $\mathcal{S}$ extracts the corrupted parties' inputs. More concretely, the simulator repetitively rewinds the adversary to the beginning of Step 4b, where for every iteration the parties evaluate their polynomial at a randomly chosen point u and the simulator extracts the individual evaluations by running the extractor of the proof of knowledge π_{EXP} and records these values only if they pass the verification phase.

   Upon recording $d+1$ values for each corrupted party, the simulator reconstructs the combined polynomials of all corrupted parties. It adds the values extracted for each party $i \in \mathcal{I}$ and uses the combined values for each of the $d+1$ points u to reconstruct a polynomial using polynomial interpolation. Then the simulator obtains the roots of the polynomials, say X', which it sets as the combined input of the corrupted parties. In other words, the roots of this polynomial can be thought of as the intersection of elements of the corrupted parties. S sends the combined input X' as the input of each corrupted party.

7. $\mathcal{S}$ receives the set intersection Z from the trusted party.

8. Finally, by running the simulator $\mathcal{S}^{\text{ML}}_{\text{DecZero}}$ on the appropriate plaintext, for every $x_1^j \in Z$, $\mathcal{S}$ enforces the decryption to be zero, and, for $x_1^j \notin Z$, the simulator sets the decryption as a random group element

The runtime analysis of the simulator in this case is the same as the previous case.

$\mathbf{Hybrid}_0$: The first game is the real execution.

$\mathbf{Hybrid}_1$: This hybrid is identical to the real world with the exception that the simulator $\mathcal{S}_1$ in this experiment extracts the corrupted parties inputs as in the simulation. More precisely, it extracts the inputs of all corrupted parties from π_{EXP}, and aborts if it fails to extract. Then, it computes X' the combined inputs of the rest of corrupted parties. It checks if the final output is correct w.r.t. X' being the inputs of the other corrupted parties. It can check this, because in this hybrid, the simulator knows the actual inputs of the honest parties. If the output is incorrect, the simulator aborts.

Indistinguishability of $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ follows similar to the same two hybrids in the previous corruption case.

$\mathbf{Hybrid}_2$: This hybrid is the same as in the previous case. The simulator invokes simulator $\mathcal{S}_{\text{GEN}}$ for protocol π_{GEN} in Step 1 and computes the set-intersection result Z based on the extracted inputs and the honest parties' inputs (which it knows in this hybrid). Next, it invokes simulator $\mathcal{S}_{\text{DecZero}}$ of the decryption protocols that is invoked in Step 4e. Note that $\mathcal{S}_{\text{DecZero}}$ is handed as plaintexts result of the set-intersection and needs to bias the outcome towards these set of plaintexts. Indistinguishability is the same as in the previous case.

$\mathbf{Hybrid}_3$: The simulator changes all the proofs given by the honest parties in Step 4b to simulated ones. It also changes the proofs given by P_1 using π_{EXP} in the first step and π_{EVAL} in Step 4d. Indistinguishability follows using the straight-line simulatability of the proofs employed.

$\mathbf{Hybrid}_4$: This hybrid is the same as in the previous case and indistinguishability follows as before.

As $\mathbf{Hybrid}_4$ is identical to the real simulator, the proof of indistinguishability follows via a standard hybrid argument. ■

4.1 An Instantiation of π_{EXP} Based on DDH in the Random Oracle Model

Our first instantiation uses the following building blocks. First, we use the El Gamal PKE as the threshold additively homomorphic encryption scheme; we elaborate in Section 2.3.3 regarding this scheme and its properties. We further consider Pedersen's commitment scheme [Ped91] for the commitment scheme made by P_1 in Step 2 (see Section 2.4 for the details of this commitment scheme). Finally we realize π_{EXP} using a standard Σ-protocol for the following relation

$$\mathcal{R}_{\text{EXP}} = \left\{((\mathbb{G}, g, h, h'), (m, r)) \mid h' = g^m h^r\right\}.$$

We invoke this proof in two places in our protocol. First, P_1 proves the knowledge of its committed input in Step 2. Next, the parties prove the knowledge of their evaluated polynomial in Step 4b (where for any El Gamal type ciphertext $\langle c_1, c_2\rangle = \langle g^r, h^r \cdot g^m\rangle$ it is sufficient to prove the knowledge with respect to the second group element c_2, which can be viewed as a Pedersen's commitment). Importantly, as the latter proof must meet the non-malleability property, we consider its non-interactive variant using the Fiat-Shamir heuristic [FS86] which is analyzed in the Random Oracle Model of Bellare and Rogaway [BR93]. Finally, we note that the overhead of this proof is constant. As mentioned before, we need the proofs to satisfy simulation-extractability. Random oracles facilitate straight-line extraction and if we assume the stronger programmability property of the random oracles, we can make the proofs straight-line simulatable. For more details, see [FKMV12].

4.2 An Instantiation of π_{EXP} Based on DLIN in the CRS Model

Our second instantiation in the CRS model is based on the [BBS04] PKE that is based on the DLIN hardness assumption and the simulation-sound NIZK by Groth [Gro06]. In this work, Groth demonstrates NIZK proofs of knowledge for Pedersen's commitment scheme, which can be used by P_1 in Step 2 as in the previous instantiation, and for a plaintext knowledge relative to [BBS04] which can be used by the parties in Step 4c. To achieve UC security we will require that an independent common reference string is sampled between every pair of parties.

4.3 Communication and Computation Complexities

Denoting by m_{MIN} (resp. m_{MAX}) the minimum (resp. maximum) over all input sets sizes and n is the number of parties, we set $m_1 = m_{\text{MIN}}$. Next, note that the communication complexity of Protocol 2 is dominated by the following factors: (1) First, $O(n^2)$ groups elements in the threshold key generation phase in Step 1, in the coin tossing generation phase in Step 4b, and in Step 4c where the parties broadcast their polynomial evaluations. (2) Second, the 2PC step for which each party P_i computes its own polynomial boils down to $O(\sum_i m_i)$, (3) the broadcast of the combined protocol and the overhead of the zero-knowledge proof π_{EVAL} yield $O(n \cdot m_{\text{MAX}} + n \cdot m_{\text{MIN}} \cdot \log m_{\text{MAX}})$ and finally, (4) the threshold decryption phase which implies $O(n \cdot m_{\text{MIN}})$ overhead. All together this implies $O((n^2 + n \cdot m_{\text{MAX}} + n \cdot m_{\text{MIN}} \cdot \log m_{\text{MAX}})\kappa)$ bits of communication. We note that the overhead in phase (4) is achieved by having each party individually communicating with P_1, raising the decrypted ciphertext to the power of a random element.

In addition to the above, except for party P_1, the computational complexity overhead of each party P_i is $O(m_{\text{MAX}})$ exponentiations plus $O(m_{\text{MIN}} \cdot m_{\text{MAX}})$ groups multiplications, whereas party P_1 needs to perform $O(m_1 \cdot m_{\text{MAX}})$ exponentiations due to the polynomial evaluations.

References

[ABKU99] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999.

[ACT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: Size-hiding private set intersection. In *PKC*, pages 156–173, 2011.

[AMP04] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k th-ranked element. In *EUROCRYPT*, pages 40–55, 2004.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

[BG13] Stephanie Bayer and Jens Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In *EUROCRYPT*, pages 646–663, 2013.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. Stoc. pages 503–513, 1990.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993.

[CF15] Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *CCS*, pages 1518–1529, 2015.

[CJS12] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. *IEICE Transactions*, 95-A(8):1366–1378, 2012.

[CKMZ14] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *CRYPTO*, pages 513–530, 2014.

[CKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, pages 213–231, 2010.

[CT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, pages 143–159, 2010.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS*, pages 789–800, 2013.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC*, pages 119–136, 2001.

[DMRY11] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Secure efficient multiparty computing of multivariate polynomials and applications. In *ACNS*, pages 130–146, 2011.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO*, pages 581–596, 2002.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 125–142, 2009.

[FHNP16] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *J. Cryptology*, 29(1):115–155, 2016.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[FJN+13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Minilego: Efficient secure two-party computation from general assumptions. In *EUROCRYPT*, pages 537–556, 2013.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *INDOCRYPT*, pages 60–79, 2012.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Gil99] Niv Gilboa. Two party RSA key generation. In *CRYPTO*, pages 116–129, 1999.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In *CCS*, pages 567–578, 2015.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[Haz15] Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In *TCC*, pages 90–120, 2015.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[HL10] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *J. Cryptology*, 23(3):422–456, 2010.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, pages 132–150, 2011.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient rsa key generation and threshold paillier in the two-party setting. In *CT-RSA*, pages 313–331, 2012.

[HN12] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *J. Cryptology*, 25(3):383–433, 2012.

[HT10] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*, pages 195–212, 2010.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, pages 577–594, 2009.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *SCN*, pages 418–435, 2010.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*, pages 818–829, 2016.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. *IACR Cryptology ePrint Archive*, 2016:505, 2016.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX*, pages 285–300, 2012.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In *EUROCRYPT*, pages 446–466, 2011.

[Lin16] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *J. Cryptology*, 29(2):456–490, 2016.

[LOP11] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The ips compiler: Optimizations, variants and concrete efficiency. In *CRYPTO*, pages 259–276, 2011.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO*, pages 319–338, 2015.

[LSS16] Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In *TCC*, 2016.

[MN15] Atsuko Miyaji and Shohei Nishida. A scalable multiparty private set intersection. In *NSS*, pages 376–385, 2015.

[MNN17] Atsuko Miyaji, Kazuhisa Nakasho, and Shohei Nishida. Privacy-preserving integration of medical data - A practical multiparty private set intersection. *J. Medical Systems*, 41(3):37:1–37:10, 2017.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO*, pages 392–404, 1991.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX*, pages 797–812, 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[RR17] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *EUROCRYPT*, pages 235–259, 2017.

[RS98] Martin Raab and Angelika Steger. "balls into bins" - A simple and tight analysis. In *RANDOM*, pages 159–170, 1998.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[SS07] Yingpeng Sang and Hong Shen. Privacy preserving set intersection protocol secure against malicious behaviors. In *PDCAT*, pages 461–468, 2007.

[SS08] Yingpeng Sang and Hong Shen. Privacy preserving set intersection based on bilinear groups. In *ACSC*, pages 47–54, 2008.

[SV15] Abhi Shelat and Muthuramakrishnan Venkitasubramaniam. Secure computation from millionaire. In *ASIACRYPT*, pages 736–757, 2015.

[Ver11] Damien Vergnaud. Efficient and secure generalized pattern matching via fast fourier transform. In *AFRICACRYPT*, pages 41–58, 2011.

[Vöc03] Berthold Vöcking. How asymmetry helps load balancing. *J. ACM*, 50(4):568–589, 2003.

[Wie07] Udi Wieder. Balanced allocations with heterogenous bins. In *SPAA*, pages 188–193, 2007.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.