Oblivious Polynomial Evaluation and Secure Set-Intersection from Algebraic PRFs[*]

Carmit Hazay[†]

Abstract

In this paper we study the two fundamental functionalities *oblivious polynomial evaluation in the exponent* and *set-intersection*, and introduce a new technique for designing efficient secure protocols for these problems (and others). Our starting point is the [BGV11] technique (CRYPTO 2011) for verifiable delegation of polynomial evaluations, using *algebraic PRFs*. We use this tool, that is useful to achieve *verifiability* in the *outsourced setting*, in order to achieve *privacy* in the *standard two-party* setting. Our results imply new simple and efficient oblivious polynomial evaluation (OPE) protocols. We further show that our OPE protocols are readily used for secure set-intersection, implying much simpler protocols in the plain model. As a side result, we demonstrate the usefulness of algebraic PRFs for various search functionalities, such as keyword search and oblivious transfer with adaptive queries. Our protocols are secure under full simulation-based definitions in the presence of malicious adversaries.

Keywords: Efficient Secure Computation, Oblivious Polynomial Evaluation, Secure Set-Intersection, Committed Oblivious PRF

[*]A preliminary version of this paper appeared in the proceedings of TCC 2015 [Haz15].

[†]Faculty of Engineering, Bar-Ilan University, Israel. Email: `carmit.hazay`@biu.ac.il. Research partially supported by a grant from the Israel Ministry of Science and Technology (grant No. 3-10883).

1 Introduction

Efficient secure two-party computation. Secure two-party computation enables two 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 (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 [ST04, IPS08, NO09, LOP11, LP11, BDOZ11, DPSZ12, NNOB12] 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. This approach has been proven useful for many different functions in both the semi-honest and malicious settings. Notable examples are calculating the kth ranked element [AMP04], pattern matching and related search problems [HT10, Ver11], set-intersection [JL09, HN12] and oblivious pseudorandom function (PRF) evaluation [FIPR05].

In this paper we study the two fundamental functionalities *oblivious polynomial evaluation in the exponent* and *set-intersection* and introduce a new technique for designing efficient secure protocols for these problems in the presence of semi-honest and malicious attacks with simulation-based security proofs. We further demonstrate that our technique is useful for various search functionalities.

Algebraic PRFs. Informally, an algebraic pseudorandom function (PRF) is a PRF with a range that forms an Abelian group such that group operations are efficiently computable. In addition, certain algebraic operations on these outputs can be computed significantly more efficiently if one possesses the key of the pseudorandom function that was used to generate them. This property is denoted by *closed form efficiency* and allows to compute a batch of l PRF values much more efficiently than by computing the l values separately and then combing them. Algebraic PRFs were exploited in [BGV11] to achieve faster verifiable polynomial evaluations (in the exponent). Specifically, in their setting, a client outsources a d-degree polynomial to an untrusted server together with some authenticating information, while the client stores a short secret key. Next, when the client provides an input for this polynomial the server computes the result and an authentication message that allows the client to verify this computation in sub-linear time in d.

More concretely, let $Q(\cdot) = (q_0, \ldots, q_d)$ be the polynomial stored on the server in the clear. Then the client additionally stores a vector of group elements $\{g^{aq_i+r_i}\}_{i=0}^{d}$ where $a \leftarrow \mathbb{Z}_p$ and p is a prime, and r_i is the ith coefficient of a polynomial $R(\cdot)$ of the same degree as $Q(\cdot)$. Then for every client's input t the server returns $y = Q(t)$ and $u = g^{aQ(t)+R(t)}$ and the client accepts u if and only if $u = g^{ay+R(t)}$. Interestingly, in case $g^{r_i} = \mathsf{PRF}_K(i)$, where PRF is an algebraic PRF, the closed form efficiency property enables the client to compute the value $g^{R(t)}$ in sub-linear time in d. Stated differently, verifiability is achieved by viewing $g^{aq_i+r_i}$ as a (one-time) message authentication code (MAC) for g^{q_i} where batch verification of multiple MACs can be computed more efficiently than verifying each MAC separately.

In this work we demonstrate the usefulness of algebraic PRFs for various two-party problems by designing secure protocols based on this primitive. In particular, we modify the way [BGV11] use algebraic

PRFs so that instead of achieving *verifiability* in the *outsourced setting*, we achieve *privacy* in the *standard two-party* setting. It is worth noting that although the main focus of [BGV11] is correctness, they do discuss how to achieve one-sided privacy by encrypting the coefficients of the polynomial. Nevertheless, it is not clear how to maintain the privacy of the input to the polynomial. In this work, we use algebraic PRFs to mask the polynomial in a different way that does not allow the verifiability of the polynomial evaluation but allows the extractability of the polynomial more easily, and demonstrate an alternative way to achieve correctness. We focus our attention on the *plain model* where no trusted setup is required.

Oblivious polynomial evaluation. The oblivious polynomial evaluation (OPE) functionality is an important functionality in the field of secure two-party computation. It considers a setting where party P_0 holds a polynomial $Q(\cdot)$ and party P_1 holds an element t, and the goal is that P_1 obtains $Q(t)$ and nothing else while P_0 learns nothing. OPE has proven to be a useful building block and can be used to solve numerous cryptographic problems; e.g., secure equality of strings, set-intersection, approximation of a Taylor series, RSA key generation, oblivious keyword search, set membership, data entanglement and more [Gil99, LP02, FNP04, FIPR05, NP06, ADDV12].

Despite its broad applicability the study of OPE was demonstrated using only few concrete secure protocols, initiated in [NP99] and further continued in [CL05, ZB05, HL09]. In particular, the only protocol with a complete simulation-based proof in the presence of malicious attacks is the protocol in [HL09]. This protocol evaluates a d-degree polynomial over a composite order group $\mathbb{Z}_N$ with $O(sd)$ modular exponentiations, where N is an RSA composite and s is a statistical security parameter.

When relying on general protocols for securely realizing the OPE functionality, the (currently most practical) approach of [DPSZ12, DKL$^+$13] for arithmetic circuits follows the preprocessing model: in an offline phase some shared randomness is generated independently of the function and the inputs; in an online phase the actual secure computation is performed. One of the main advantages of these protocols is that the basic operations are almost as cheap as those used in the passively secure protocols. To get good performance, these protocols use the somewhat-homomorphic SIMD approach that handles many values in parallel in a single ciphertext, and thus more applicable for large degree polynomials. Similarly, protocols for Boolean circuits apply the cut-and-choose technique which requires to repeat the computation s times in order to prevent cheating except with probability 2^{-s} [Lin13].

In some applications such as password-based authenticate key exchange protocols or when sampling an element from a d-wise independence space, the polynomial degree is typically small and even a constant. In these cases, our protocols have clear benefits since they are much simpler, efficient and easily implementable.

Secure set-intersection. In the set-intersection problem parties P_0, P_1, holding input sets X, Y of sizes m_X and m_Y, respectively, wish to compute $X \cap Y$. 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; see [FNP04, KS05, DSMRY09, JL09, JL10, HL10a, HN12] for a few examples. For instance, 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 solve this problem securely in the plain model: (**1**) oblivious polynomial evaluation and (**2**) committed oblivious PRF evaluation. In the former approach party P_0 computes a polynomial $Q(\cdot)$ such that $Q(x) = 0$ for all $x \in X$. This polynomial is then encrypted using homomorphic encryption and sent to P_1, that computes the encryption of $r_y \cdot Q(y) + y$ for all $y \in Y$, and using fresh randomness r_y. Upon receiving these ciphertexts, P_0 decrypts them and outputs the intersection of X and these plaintexts. This approach (or a variant of it) was taken in [FNP04, KS05, DSMRY09, HN12].

To move towards malicious security, Hazay and Nissim modified this protocol so that the payload information obtained by P_1 is a random string rather than y. The main challenge in their protocol is with ensuring consistency between the correct payload information and the corresponding input element.

The second approach uses a secure implementation of oblivious pseudorandom function evaluation. Namely, P_0 chooses a PRF key K and computes the set $\mathsf{PRF}_X = \{\mathsf{PRF}_K(x)\}_{x\in X}$. The parties then execute an oblivious PRF protocol where P_0 inputs K, whereas P_1 inputs the set Y and learns the set $\mathsf{PRF}_Y = \{\mathsf{PRF}_K(y)\}_{y\in Y}$. Finally, P_0 sends the set PRF_X to P_1 that computes $\mathsf{PRF}_X \cap \mathsf{PRF}_Y$ and extracts the actual intersection. This idea was introduced in [FIPR05] and further used in [HL10a, JL09, JL10]. Other solutions in the random oracle model [CT10, CKT10, ACT10, DCW13, PSSZ15] take a different approach by applying the random oracle on (one of) the sets members or apply oblivious transfer extension.

In a recent result [PSZ14], Pinkas et al. compare the overhead of exiting solutions for set-intersection in the semi-honest setting. One of their conclusions is that OPE-based protocols are inferior to protocols that rely on oblivious-transfer extension.

To the best of our knowledge, the most efficient protocol in the malicious plain model that does not require a trusted setup or rely on non-standard assumptions is the protocol of [HN12] that incurs computation of $O(m_X + m_Y \log(m_X + m_Y))$ modular exponentiations. A more efficient protocol with $O(m_X + m_Y)$ communication and computational costs was introduced by [JL09] in the common reference string (CRS) model (where the CRS includes a safe RSA composite that determines the group order and implies high overhead when produced using a secure two-party protocol). Another drawback of this protocol is that its security proof runs an exhaustive search on the input domain of the PRF in order to extract P_0's input. This implies that the proof only works for small domain PRFs and that the complexity of the simulator grows linearly with the size of the PRF's input domain.

Committed oblivious PRF evaluation. The oblivious PRF evaluation functionality $\mathcal{F}_{\mathrm{PRF}}$ that *obliviously* evaluates a PRF is defined by $(K, x) \mapsto (-, \mathsf{PRF}_K(x))$. This functionality is very important in the context of secure computation since it essentially implements a random oracle. That is, the party with the PRF key, say P_0, mimics the random oracle role via interaction. Therefore, if the protocol that realizes $\mathcal{F}_{\mathrm{PRF}}$ is simulation-based secure then both desirable properties of a random oracle, *programmability* and *observability*, can be achieved by this protocol. First, since the simulator can force any output for a corrupted P_1, it essentially programs the function's output. In addition, it can also observe (via extraction) the input to the functionality. Nevertheless, the usefulness of oblivious PRF evaluation is reflected via an additional property of a committed key that implies that the *same key* is used for multiple PRF evaluations.

Committed oblivious PRF (CPRF) evaluation has been used to compute secure set-intersection [JL09, HL10a], oblivious transfer with adaptive queries [FIPR05], keyword search [FIPR05], pattern matching [HL10a, FHV13] and more. It is therefore highly important to design efficient protocols for this functionality. Current implementations of the [NR97] algebraic PRF, discussed in this paper, employ an oblivious transfer protocol for each input bit [FIPR05, HL10a] and are only secure for a single PRF evaluation. Consequently, the protocol of [HL10a] does not achieve full security against malicious adversaries. In addition, the protocol from [JL09] (that implements a variant of the [DY05] PRF) requires a trusted setup of a safe RSA composite and suffers from the drawbacks specified above.

1.1 Our Results

In this paper we use algebraic PRFs to design alternative simple and efficient protocols for polynomial evaluation, set-intersection, committed oblivious PRF evaluation and search problems. Our protocols are based on a new approach that provides simplicity and modularity, as well as asymptotic and concrete improvement. Below, we demonstrate the broad usefulness of our technique for the different functionalities.

Reference	Hardness Assumption	Overhead (Number of Exp.)	
[HL09]	DCR	$O(sd)$, where s is stat. parameter	
This Work	d-strong DDH	$2d + O(1)$	In the exponent
This Work	DDH	$2d + O(\log d)$	In the exponenet

Table 1: Comparisons with a prior OPE construction, where d is the polynomial degree and DCR is the underlying hardness assumption for the Paillier encryption scheme [Pai99].

Oblivious polynomial evaluation (Section 3). Our first construction realizes in the plain model the OPE functionality *in the exponent*, with simulation-based security against semi-honest and malicious attacks. In more details, we use algebraic PRFs to build simple two-phases OPE protocols as follows. In the first phase party P_0, holding the polynomial $g^{Q(\cdot)}$, publishes its masked polynomial $g^{Q(\cdot)+R(\cdot)}$ where the set $g^{R(\cdot)}$ is determined by an algebraic PRF. Next, P_1 locally computes $g^{Q(t)+R(t)}$ and the parties run an unmasking phase for obliviously evaluating $g^{R(t)}$ for P_1, where the overhead of the latter phase is dominated by the overhead of the closed form efficiency property of the specific PRF; see Table 1.

In this work, we consider two PRF implementations used by [BGV11]: (**1**) a PRF with security under the strong-DDH assumption.[1] (**2**) The Naor-Reingold PRF [NR97] with security under the DDH assumption. More concretely, the efficiency of our protocols is only $d + 1$ modular exponentiations for the first phase of sending the masked polynomial, and $d + 1 + O(1)$ (resp. $O(\log d)$) modular exponentiations for the second phase of obliviously evaluating the pseudorandom polynomial under the strong-DDH (resp. DDH) assumption. For simplicity, we only consider univariate polynomials. Nevertheless, our technique can be applied to multivariate polynomials as well (with total degree d or of degree d in each variable); see [BGV11] for further details. To the best of our knowledge, our protocols are the first to obliviously evaluate both univariate and multivariate polynomials that efficiently. Finally, we stress that evaluating a polynomial in the exponent as we demonstrate in this work, has strong applicability in the context of set membership, where the goal is to privately verify membership in some secret set, as well as achieving d-wise independence.

Secure set-intersection (Section 4). Next, we demonstrate that algebraic PRFs are useful for designing new and simple protocols for set-intersection according to both approaches of OPE and committed oblivious PRF considered above. We first show that our protocols for OPE readily induce new protocols for set-intersection. That is, P_0 first encodes the set X by a polynomial $g^{Q(\cdot)}$ as specified above, and masks it. Next, for each $y \in Y$ party P_1 verifies whether the masked polynomial evaluation of y equals the evaluation $g^{R(y)}$, and concludes whether the element is in the intersection. We note that our approach greatly simplifies the use of OPE for set-intersection protocols, as now no payload is transferred back to P_0. Note also that this naive approach requires a multiplicative overhead (in the sets sizes) since for each element in its input Y, P_1 needs to evaluate a polynomial of degree m_X. To reduce the computational overhead we additionally employ various hash functions schemes that split the elements into smaller bins. In this case, the elements mapped by P_0 to a certain bin must only be compared to those mapped by P_1 to the same bin. This approach was taken by Freedman et al. in [FNP04] in the semi-honest, and further extended to the malicious setting by Hazay and Nissim [HN12], which introduced a fairly complicated protocol, as it requires the combination of both approaches of OPE and oblivious PRF evaluation.

We introduce the hashing technique into our constructions and provide a generic description that can be

[1]Where the dynamic d-strong assumption asserts, informally speaking, that it is hard to distinguish a tuple $\left(g^x, g^{x^2}, \ldots, g^{x^d}\right)$ from $\left(g^{x_1}, g^{x_2}, \ldots, g^{x_d}\right)$ for random choices of $x, x_1, \ldots, x_d$.

Reference	Modeling	Hardness Assumption	Overhead (Number of Exp.)
[JL09]	**CRS of a safe prime**	**Decisional d-DHI**	$\mathbf{O(m_X + m_Y)}$
[HN12]	**plain model**	**DDH**	$\mathbf{O(m_X + m_Y \log (m_X + m_Y))}$
[DCW13]	random oracle	random oracle	$O(n)$, where n is sec. parameter
This Work – OPE	plain model	d-strong DDH	$O(m_X + m_Y \log\log m_X)$
This Work – OPE	**plain model**	**DDH**	$\mathbf{O(m_X + m_Y \log m_X)}$
This Work – CPRF	**plain model**	**d-strong DDH**	$\mathbf{O(m_X + m_Y)}$
This Work – CPRF	plain model	DDH	$O((m_X + m_Y)\log(m_X + m_Y))$

Table 2: Comparisons with prior set-intersection constructions, where m_X (resp. m_Y) is the size of P_0's (resp. P_1's) input X (resp. Y). We highlight the protocols with the best performance under each assumption.

instantiated with different hash functions. Our protocols are far less complicated and maintain a modular description. Specifically, we devise an alternative zero-knowledge proof for verifying the correctness of the hashed polynomials while exploiting the algebraic properties of the PRF. Under the strong-DDH assumption our protocol matches the communication overhead of the protocol from [JL09] (that also relies on a dynamic hardness assumption) and implies the computation of $O(m_X + m_Y \log\log m_X)$ exponentiations, with the benefits that it operates over prime order groups and it does not require a trusted setup. Under the DDH assumption our protocol, using hash functions implies the computation of $O(m_X + m_Y \log m_X)$ exponentiations. We note that this overhead does not asymptotically improve the overhead of [HN12] but greatly simplifies its description by, for instance, avoiding using costly PRF evaluations.

Next, we show that algebraic PRFs are useful for applications that rely on committed oblivious PRF evaluation. Our results for set-intersection are summarized in Table 2.

Committed oblivious PRF evaluation (Section 5). Observing that the batch computation for l PRF values $\mathsf{PRF}'_K(x) = \prod_{i=0}^{l}[\mathsf{PRF}_K(i)]^{x^i}$ is a PRF as well,[2] we derive new PRF constructions in prime order groups and more interestingly, simple committed oblivious PRF evaluation protocols. Our strong-DDH based PRF requires constant overhead, and our DDH-based protocol is the simplest committed oblivious PRF implementation for the [NR97] function; see table 3.

We use our new protocols to realize set-intersection, considering the second approach of committed oblivious PRF evaluation. Plugging-in our protocols for committed oblivious PRF evaluation in the protocols cited above requires some adjustments, but implies malicious security fairly immediately. Specifically, a subtle point arises as P_0 must generate its PRF key *independently* of the inputs, since otherwise it may enforce collisions on elements from X and Y, causing the simulation to fail. We note that this subtlety does not arise in [JL09] due to the fact that the PRF is only defined on a specific bounded domain. Our protocols using committed oblivious PRF imply set-intersection protocols with $O(m_X + m_Y)$ costs under the strong-DDH assumption and $((m_X + m_Y)\log(m_X + m_Y))$ communication and computation costs under the DDH assumption, where the former analysis matches the overhead from [JL09].

[2]Specifically, it is a PRF when l is a polynomial and therefore only results in a PRF with polynomial input domain size.

Reference	Modeling	Hardness Assumption	Overhead (Number of Exp.)
[FIPR05, HL10a]	plain model	DDH	$O(t)$ prime order groups
[JL09] (committed)	CRS of a safe RSA composite	q-DHI	$O(1)$ composite order groups
This Work (committed)	plain model	DDH	$O(t)$ prime order groups
This Work (committed)	plain model	d-strong DDH	$O(1)$ prime order groups

Table 3: Comparisons with prior (committed) oblivious PRF constructions, where t is the input bit-length. Furthermore, the [JL09] proof employs an exhaustive search on the input domain.

2 Preliminaries

2.1 Basic Notations

We denote the security parameter by n. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large n it holds that $\mu(n) < \frac{1}{p(n)}$. 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 $X = \{X(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ and $Y = \{Y(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ be two distribution ensembles. We say that X and Y are* computationally indistinguishable*, denoted $X \overset{\text{c}}{\approx} Y$, if for every PPT machine D, every $a \in \{0,1\}^*$, every positive polynomial $p(\cdot)$ and all sufficiently large n:*

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

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

All our constructions rely on the following hardness assumptions.

DECISIONAL DIFFIE-HELLMAN. The decisional Diffie-Hellman assumption is stated as follows.

Definition 2.2 (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*

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

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

STRONG DIFFIE-HELLMAN. The strong Diffie-Hellman family of assumptions is parameterized by a parameter d, and states that any PPT distinguisher cannot distinguish elements $g, g^x, g^{x^2}, \ldots, g^{x^d}$ from a sequence of random elements in a prime order group $\mathbb{G}$. More formally,

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

$$\left| \Pr\left[D(\mathbb{G}, p, g, g^x, g^{x^2}, \ldots, g^{x^d}) = 1\right] - \Pr\left[D(\mathbb{G}, p, g, g^{x_1}, g^{x_2}, \ldots, g^{x_d}) = 1\right] \right| \leq \mathsf{negl}(n),$$

where $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^n)$ and the probabilities are taken over the choices of $x, x_1, \ldots, x_d \leftarrow_R \mathbb{Z}_p$.

We refer the reader to [BGV11] for further discussions and related work regarding the strong Diffie-Hellman assumption. Fixing the parameter d appropriately enables [BGV11] to evaluate a special class of polynomials of degree d in constant time. Specifically, there is a tight correlation between the value of d and the efficiency of their construction; the stronger the assumption is, the larger the class of polynomials that can be evaluated in constant time. In what follows, we also fix the parameter d in this assumption according to the degree of evaluated polynomial.

2.3 Public Key Encryption (PKE)

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

Definition 2.4 (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 n (in unary), outputs keys* $(\mathrm{PK}, \mathrm{SK})$*, where* PK *is a public key and* SK *is a secret key. We denote this by* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^n)$.
- 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*:

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

Denote by $\mathbf{Adv}_{\Pi,\mathcal{A}}(n)$ the probability that $\mathcal{A}$ wins the IND-CPA game.

Definition 2.5 (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* $\mathbf{Adv}_{\Pi,\mathcal{A}}(n) \leq \frac{1}{2} + \mathsf{negl}(n)$.

2.3.1 Additively Homomorphic PKE

A public key encryption scheme is additively homomorphic if given two ciphertexts $c_1 = \mathsf{Enc}_{\mathrm{PK}}(m_1)$ and $c_2 = \mathsf{Enc}_{\mathrm{PK}}(m_2)$ it is possible to efficiently compute $\mathsf{Enc}_{\mathrm{PK}}(m_1 + m_2; r)$ 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 2.6 (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* $\mathsf{Gen}(1^n)$*, 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 $\mathcal{C}$.*[3]
- *For every $m_1, m_2 \in \mathcal{M}$ it holds that*

$$\mathsf{Dec}_{\mathrm{SK}}\big(\mathsf{Enc}_{\mathrm{PK}}(m_1) \cdot \mathsf{Enc}_{\mathrm{PK}}(m_2)\big) = m_1 + m_1$$

  with overwhelming probability, where the group operations are carried out in $\mathcal{C}$ and $\mathcal{M}$, respectively.

Note that any such a scheme supports a multiplication of a plaintext by a scalar.

2.4 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 $\mathrm{PK} = \langle \mathbb{G}, p, g, h \rangle$ and the corresponding secret key is $\mathrm{SK} = s$, s.t. $g^s = h$. Encryption is performed by choosing $r \leftarrow \mathbb{Z}_p$ and computing $\mathsf{Enc}_{\mathrm{PK}}(m; r) = \langle g^r, h^r \cdot g^m \rangle$. Decryption of a ciphertext $C = \langle c_1, c_2 \rangle$ is performed by computing $g^m = c_2 \cdot c_1^{-s}$ and then finding m by running an exhaustive search.

2.4.1 Distributed El Gamal

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. We consider two functionalities: One for securely generating a secret key while keeping it a secret from both parties, whereas the second functionality jointly decrypts a given ciphertext. We denote the key generation functionality by $\mathcal{F}_{\mathrm{KEY}}$, which is defined as follows,

$$(1^n, 1^n) \mapsto \Big((\mathrm{PK}, \mathrm{SK}_1), (\mathrm{PK}, \mathrm{SK}_2)\Big)$$

where $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^n)$, and SK_1 and SK_2 are random shares of SK. The decryption functionality $\mathcal{F}_{\mathrm{DEC}}$ is defined by,

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

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 other. Finally, the parties compute $h = h_1 h_2$ and set $\mathrm{PK} = \langle \mathbb{G}, p, g, h \rangle$.

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

Clearly, the secret key $s = s_1 + s_2$ 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 $\pi_{\rm DL}$, specified in Section 2.5. To ensure simulation based security, P_1 must commit to its share first and decommit this commitment after P_2 sends its share. Moreover, decryption of a ciphertext $c = \langle c_1, c_2 \rangle$ follows by computing $c_2 \cdot (c_1^{x_1} \cdot c_1^{x_2})^{-1}$, where each party sends c_i to the power of its share together with a proof. We denote these protocols by $\pi_{\rm KEY}$ and $\pi_{\rm DEC}$, respectively, and assume that they can be carried out with simulation-based security in the presence of malicious attacks. A variation of $\pi_{\rm DEC}$ allows the parties to learn whether a ciphertext $c = \langle c_1, c_2 \rangle$ 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). They then decrypt the final ciphertext and conclude that $m = 0$ if and only if the masked plaintext was 0. We denote this protocol by $\pi_{{\rm DEC}_0}$.

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. Our proofs are Σ-protocols with a constant overhead. A generic efficient technique that enables to transform any Σ-protocol into a zero-knowledge proof of knowledge can be found in [HL10b]. This transformation requires additional 6 exponentiations.

1. $\pi_{\rm DL}$, for demonstrating the knowledge of a solution x to a discrete logarithm problem [Sch89].

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

2. $\pi_{\rm 2DL}$, for demonstrating the knowledge of a solution x, y to a discrete logarithm problem [Oka92].

$$\mathcal{R}_{\rm 2DL} = \{((\mathbb{G}, g, h, z), (x, y)) \mid z = g^x h^y\}.$$

3. $\pi_{\rm DDH}$, for demonstrating that an El Gamal ciphertext is an encryption of zero [CP92].

$$\mathcal{R}_{\rm DDH} = \{((\mathbb{G}, g, h, g_1, h_1), x) \mid g_1 = g^x \wedge h_1 = h^x\}.$$

4. $\pi_{\rm Eq-Exp}$, for proving that two ciphertexts under two different public-keys encrypt the same plaintext, where the plaintext lies in the exponent of two different groups. This boils down to proving equality of exponentiations over the integers, without utilizing the group description.

$$\mathcal{R}_{\rm Eq-Exp} = \left\{((\mathrm{PK}_1, \mathrm{PK}_2, c_1, c_2), (\alpha, r_1, r_2)) \mid c_1 = \mathsf{Enc}_{\rm PK}(\alpha) \wedge c_2 = \mathsf{Enc}'_{\rm PK'}(\alpha)\right\}.$$

In our case, we need to prove consistency between El Gamal PKE and an additively homomorphic encryption scheme (e.g., Paillier). A discussion about this proof can be found in [HMRT12].

5. $\pi_{\rm MULT}$, for proving that a ciphertext c_2 encrypts a product of two plaintexts values. Namely,

$$\mathcal{R}_{\rm MULT} = \left\{((\mathbb{G}, \mathrm{PK}, c_0, c_1, c_2), (a_0, a_1, r_0, r_1, r_2,)) \mid \begin{array}{l} c_i = \mathsf{Enc}_{\rm PK}(a_i; r_i) \text{ for } i \in \{0,1\} \wedge \\ c_2 = \mathsf{Enc}_{\rm PK}(a_0 \cdot a_1; r_2) \end{array}\right\}$$

where multiplication is performed in the corresponding plaintext group. A zero-knowledge proof for the El Gamal PKE, that is based on the Damgård, Jurik and Nielsen technique [DJN10], can be found in [HN12].

6. π_{EQ}, for demonstrating equality of two exponentiations in the same group. Namely,

$$\mathcal{R}_{\mathrm{EQ}} = \left\{((\mathrm{PK}, c_1, c_1', c_2, c_2'), (m, r_1, r_2)) \mid c_1' = c_1^m \cdot \mathsf{Enc}_{\mathrm{PK}}(0; r_1) \wedge c_2' = c_2^m \cdot \mathsf{Enc}_{\mathrm{PK}}(0; r_2)\right\}.$$

A variant of this zero-knowledge proof was presented and discussed in [HMRT12] for Paillier encryption scheme and can be easily extended for this relation as well. We specify the details of this proof for the El Gamal encryption scheme (in a honest verifier format).

Protocol 1 (Zero-knowledge proof of knowledge for $\mathcal{R}_{\mathrm{EQ}}$):

- **Joint statement:** *Public key* PK *and ciphertexts* (c_1, c_1', c_2, c_2') *over group* $\mathbb{G}$ *of prime order* p.
- **Auxiliary input for the prover:** *Elements* (m, r_1, r_2) *such that the conditions within* $\mathcal{R}_{\mathrm{EQ}}$ *are met.*
- **The protocol:**
  - *(a) P chooses at random* $t, s_1, s_2 \leftarrow \mathbb{Z}_p$ *and sets* $\tilde{c}_1 = c_1^t \cdot \mathsf{Enc}_{\mathrm{PK}}(0; s_1)$ *and* $\tilde{c}_2 = c_2^t \cdot \mathsf{Enc}_{\mathrm{PK}}(0; s_2)$. *That is, let* $c_1 = \langle c_1^1, c_1^2\rangle$ *and* $c_2 = \langle c_2^1, c_2^2\rangle$, *then* $\tilde{c}_1 = \langle (c_1^1)^t \cdot g^{s_1}, (c_1^2)^t \cdot h^{s_1}\rangle$ *and* $\tilde{c}_2 = \langle (c_2^1)^t \cdot g^{s_2}, (c_2^2)^t \cdot h^{s_2}\rangle$. *P sends V the values* $\tilde{c}_1 = \langle \tilde{c}_1^1, \tilde{c}_1^2\rangle, \tilde{c}_2 = \langle \tilde{c}_2^1, \tilde{c}_2^2\rangle$.
  - *(b) V chooses* $c \leftarrow \mathbb{Z}_p$ *at random and sends it to P.*
  - *(c) P responds with* $cm + t$, $cr_1 + s_1$ *and* $cr_2 + s_2$.
  - *(d) Upon receiving* z_1, z_2 *and* z_3, *V accepts if*
    - $\langle (c_1^1)^{z_1} \cdot g^{z_2}, (c_1^2)^{z_1} \cdot h^{z_2}\rangle = \langle (c_1'^1)^c \cdot \tilde{c}_1^1, (c_1'^2)^c \cdot \tilde{c}_1^2\rangle$ *and*
    - $\langle (c_2^1)^{z_1} \cdot g^{z_3}, (c_2^2)^{z_1} \cdot h^{z_3}\rangle = \langle (c_2'^1)^c \cdot \tilde{c}_2^1, (c_2'^2)^c \cdot \tilde{c}_2^2\rangle$,

    where $c_1' = \langle c_1'^1, c_1'^2\rangle$ *and* $c_2' = \langle c_2'^1, c_2'^2\rangle$.

Proposition 2.1 *Protocol* 1 *is a zero-knowledge honest verifier proof for* $\mathcal{R}_{\mathrm{EQ}}$.

Proof: Completeness is easy to verify since the conditions that the verifier checks are always met. Next, we prove zero-knowledge by designing a simulator $\mathcal{S}_{\mathrm{EQ}}$ as follows. $\mathcal{S}_{\mathrm{EQ}}$ fixes the last message $z_1, z_2, z_3 \leftarrow \mathbb{Z}_p$ at random and sets

$$\tilde{c}_1^1 = (c_1^1)^{z_1} \cdot g^{z_2}/(c_1'^1)^c, \quad \tilde{c}_1^2 = (c_1^2)^{z_1} \cdot h^{z_2}/(c_1'^2)^c$$
$$\tilde{c}_2^1 = (c_2^1)^{z_1} \cdot g^{z_3}/(c_2'^1)^c, \quad \tilde{c}_2^2 = (c_2^2)^{z_1} \cdot h^{z_3}/(c_2'^2)^c.$$

Note that the simulated $\tilde{c}_1^1$ equals $(c_1^1)^{z_1 - mc} \cdot g^{z_2 - r_1 c}$ since $c_1'^1 = (c_1^1)^m \cdot g^{r_1}$. A similar argument can be made for $\tilde{c}_1^2$ and $\tilde{c}_2$. Finally, soundness holds due to the fact that the proof is a proof of knowledge as well, where two views with the same pair of messages $(\tilde{c}_1, \tilde{c}_2)$ and two challenges c, c' enable to extract m, r_1, r_2. ■

3 Protocols for Oblivious Polynomial Evaluation

In this section we introduce our new constructions for oblivious polynomial evaluation (OPE) in the exponent, implementing functionality $\mathcal{F}_{\mathrm{OPE}} : (g^{Q(\cdot)}, t) \mapsto (-, g^{Q(t)})$ for $Q(\cdot) = (q_0, \ldots, q_d)$. In particular, we assume common knowledge of the public parameters: a multiplicative group $\mathbb{G}$ of order p and a generator g for $\mathbb{G}$, and that the polynomial coefficients are in $\mathbb{Z}_p$. In our solution, party P_0 generates these parameters and publishes its masked polynomial $g^{Q(\cdot)+R(\cdot)}$, where the set of values $g^{R(\cdot)}$ is determined by an algebraic PRF that has a closed form efficient computation for univariate polynomials (see Section 3.1). Next, P_1 computes $g^{Q(t)+R(t)}$ and the parties run an unmasking secure computation for obliviously evaluating $g^{R(t)}$ for P_1. Importantly, the closed form efficiency property of the PRF allows the parties to mutually compute $g^{R(t)}$ in *sub-linear* time in d. Before presenting our OPE constructions we formally define algebraic pseudorandom functions.

3.1 Algebraic Pseudorandom Functions [BGV11]

Algebraic PRFs are PRFs with two additional algebraic properties. First, they map their inputs into some Abelian group, where certain algebraic operations on these outputs can be computed signicantly faster if one possesses the PRF key. These properties were exploited in [BGV11] to achieve faster polynomial evaluations (in the exponent), where the coefficients of these polynomials lie in the PRF range. Several constructions, implying different overheads, were introduced in [BGV11]; we focus our attention on their constructions for univariate polynomials. Our protocols can be applied for multivariate polynomials as well (with total degree d or of degree d in each variable). We begin with the formal definition of algebraic PRFs.

Definition 3.1 (Algebraic PRFs) *We say that* $\mathrm{PRF} = (\mathsf{KeyGen}, \mathsf{PRF}, \mathsf{CFEval})$, *is an algebraic PRF if* $\mathsf{KeyGen}, \mathsf{PRF}$ *are polynomial-time algorithms specified as follows:*

- KeyGen, *given a security parameter* 1^n, *and a parameter* $m \in \mathbb{N}$ *that determines the domain size of the PRF, outputs a pair* $(K, param) \leftarrow \mathcal{K}_n$, *where* $\mathcal{K}_n$ *is the key space for a security parameter* n. K *is the secret key of the PRF, and* $param$ *encodes the public parameters.*

- PRF, *given a key* K, *public parameters* $param$, *and an input* $x \in \{0,1\}^m$, *outputs a value* $y \in Y$, *where* Y *is some set determined by* $param$.

- *In addition, the following properties hold:*

  Pseudorandomness. *We say that* PRF *is pseudorandom if for every PPT adversary* $\mathcal{A}$, *and every polynomial* $m = m(n)$, *there exists a negligible function* negl *such that*

$$|\Pr[\mathcal{A}^{\mathsf{PRF}_K(\cdot)}(1^n, param) = 1] - \Pr[\mathcal{A}^{f_n(\cdot)}(1^n, param) = 1]| \leq \mathsf{negl}(n),$$

  where $(K, param) \leftarrow \mathsf{KeyGen}(1^n, m)$ *and* $f_n : \{0,1\}^m \mapsto Y$ *is a random function.*

  Algebraic. *We say that* PRF *is algebraic if the range* Y *of* $\mathsf{PRF}_K(\cdot)$ *for every* $n \in \mathbb{N}$ *and* $(K, param) \leftarrow \mathcal{K}_n$ *forms an Abelian multiplicative group. We require that the group operation on* Y *be efficiently computable given* $param$.

  Closed form efficiency. *Let* N *be the order of the range sets of* PRF *for security parameter* n. *Let* $z = (z_1, \ldots, z_l) \in (\{0,1\}^m)^l$, $k \in \mathbb{N}$, *and efficiently computable* $h : \mathbb{Z}_N^k \mapsto \mathbb{Z}_N^l$ *with* $h(x) = \langle h_1(x), \ldots, h_l(x) \rangle$. *We say that* (h, z) *is closed form efficient for PRF if there exists an algorithm* $\mathsf{CFEval}_{h,z}$ *such that for every* $x \in \mathbb{Z}_N^k$,

$$\mathsf{CFEval}_{h,z}(x, K) = \prod_{i=1}^{l} [\mathsf{PRF}_K(z_i)]^{h_i(x)}$$

  and the running time of CFEval *is polynomial in* n, m, k *but sublinear in* l.

The last property is very important for our purposes since it allows to run certain computations very fast when the secret key is known. We next describe two implementations for algebraic PRFs introduced in [BGV11].

3.1.1 Algebraic PRFs From Strong DDH

Let $\mathcal{G}$ be a computational group scheme. The following construction PRF_1 is an algebraic PRF with polynomial sized domains.

$\mathsf{KeyGen}(1^n, m)$: Generate a group description $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^n)$. Choose $k_0, k_1 \leftarrow \mathbb{Z}_p$. Output $param = (m, p, g, \mathbb{G}), K = (k_0, k_1)$.

$\mathsf{PRF}_K(x)$: Interpret x as an integer in $\{0, \dots, D = 2^m\}$ where D is polynomial in n. Compute and output $g^{k_0 k_1^x}$.

Closed form efficiency for polynomials of degree d. We now show an efficient closed form for PRF_1 for polynomials of the form (where evaluation is computed in the exponent)

$$Q(x) = \mathsf{PRF}_K(0) \cdot \mathsf{PRF}_K(1)^x \cdot \ldots \cdot \mathsf{PRF}_K(d)^{x^d} = \prod_{i=0}^{d} \mathsf{PRF}_K(i)^{x^i}$$

where $d \leq D$. Let $h : \mathbb{Z}_p \mapsto \mathbb{Z}_p^{d+1}$, be defined as $h(x) \overset{\text{def}}{=} (1, x, \dots, x^d)$ and $(z_0, \dots, z_d) = (0, \dots, d)$. Then, we can define

$$\mathsf{CFEval}_h(x, K) \overset{\text{def}}{=} g^{\frac{k_0(k_1^{d+1}x^{d+1}-1)}{k_1 x - 1}}.$$

Specifically, we write

$$\prod_{1=0}^{d} [\mathsf{PRF}_K(z_i)]^{h_i(x)} = \prod_{i=0}^{d} [g^{k_0 k_1^i}]^{x^i} = g^{k_0 \sum_{i=0}^{d} k_1^i x^i}.$$

Correctness of CFEval follows by the identity $\sum_{i=0}^{d} k_0 k_1^i x^i = \frac{k_0((k_1 x)^{d+1}-1)}{k_1 x - 1}$.

Theorem 3.2 ([BGV11]) *Suppose that the D-Strong DDH assumption holds. Then,* PRF_1 *is a pseudorandom function.*

3.1.2 Algebraic PRFs From DDH

Let $\mathcal{G}$ be a computational group scheme. Define PRF_2 as follows.

$\mathsf{KeyGen}(1^n, m)$: Generate a group description $(p, g, \mathbb{G}) \leftarrow \mathcal{G}(1^n)$. Choose $k_0, k_1, \dots, k_m \leftarrow \mathbb{Z}_p$. Output $param = (m, p, g, \mathbb{G}), K = (k_0, k_1, \dots k_m)$.

$\mathsf{PRF}_K(x)$: Interpret $x = (x_1, \dots, x_m)$ as an m-bit string. Compute and output $g^{k_0 \prod_{i=1}^{m} k_i^{x_i}}$.

This function is known by the Naor-Reingold function [NR97].

Closed form for polynomials of degree d. We describe an efficient closed form for PRF_2 for computing polynomials of the same form as above. That is,

$$Q(x) = \mathsf{PRF}_K(0) \cdot \mathsf{PRF}_K(1)^x \cdot \ldots \cdot \mathsf{PRF}_K(d)^{x^d} = \prod_{i=0}^{d} \mathsf{PRF}_K(i)^{x^i}.$$

Let $h : \mathbb{Z}_p \mapsto \mathbb{Z}_p^{d+1}$, defined as $h(x) = (1, x, \dots, x^d)$ and let $z = (z_1, \dots, z_l) = (0, \dots, d)$ then

$$\mathsf{CFEval}_{h,z}(x, K) \overset{\text{def}}{=} g^{k_0(1+k_1 x)(1+k_2 x^2)\dots(1+k_m x^{2^m})}$$

with $m = \lceil \log d \rceil$ (clearly, d must be a power of 2).

Theorem 3.3 ([NR97]) *Suppose that the DDH assumption holds. Then,* PRF_2 *is a pseudorandom function.*

To this end, we only consider $z = (0, \dots, d)$ and omit z from the subscript, writing $\mathsf{CFEval}_h(x, K)$ instead.

3.2 Our OPE Constructions

We describe our protocol for oblivious polynomial evaluation in the $\mathcal{F}_{\mathrm{MaskPoly}}$-hybrid setting, where the parties have access to a trusted party that computes functionality $\mathcal{F}_{\mathrm{MaskPoly}} : (K, t) \mapsto (-, g^{R(t)})$ relative to some prime order group $\mathbb{G}$ and generator g that are picked by P_0, for $g^{R(\cdot)} = \big(g^{r_0}, \ldots, g^{r_d}\big)$ and $g^{r_i} = \mathsf{PRF}_K(i)$ for all i. For simplicity, we first describe a semi-honest variant of our protocol and then show how to enhance its security into the malicious setting. Formally, let $\mathrm{PRF} = \langle \mathsf{KeyGen}, \mathsf{PRF}, \mathsf{CFEval} \rangle$ denote an algebraic PRF with a range group $\mathbb{G}$ (cf. Definition 3.1), then our semi-honest protocol follows.

Protocol 2 (Protocol π_{OPE} with semi-honest security.)

- **Input:** *Party P_0 is given a d-degree polynomial $g^{Q(\cdot)} = (g^{q_0}, \ldots, g^{q_d})$ with coefficients q_i's from $\mathbb{Z}_p$ with respect to prime order group $\mathbb{G}$ and generator g. Party P_1 is given an element t from $\mathbb{Z}_p$. Both parties are given a security parameter 1^n, group description $\mathbb{G}, p$ and g.*
- **The protocol:**
  1. **Masking the polynomial.** *P_0 invokes $(K, param) \leftarrow \mathsf{KeyGen}(1^n, \lceil \log d \rceil)$ where $param$ includes a group description $\mathbb{G}$ of prime order p and a generator g. It next defines a sequence of d elements $\widetilde{R}(\cdot) = (\tilde{r}_0, \ldots, \tilde{r}_d)$ over $\mathbb{G}$ where $\tilde{r}_i = \mathsf{PRF}_K(i)$ for all i.*
     P_0 sends P_1 $param$ and the masked polynomial $C(\cdot) = \big(g^{q_0}\tilde{r}_0, \ldots, g^{q_d}\tilde{r}_d\big)$, where multiplication is implemented (componentwise) in $\mathbb{G}$.
  2. **Unmasking the result.** *Upon receiving the masked polynomial $C(\cdot) = (c_0, \ldots, c_d)$, party P_1 computes the polynomial evaluation $C(t) = \prod_{i=0}^{d} (c_i)^{t^i}$. I.e., $C(\cdot)$ is evaluated* in the exponent. *Next, the parties invoke an ideal execution of $\mathcal{F}_{\mathrm{MaskPoly}}$ where the input of P_0 is K and the input of P_1 is t. Let Z denote the output of P_1 from this ideal call, then P_1's output is $C(t)/Z$ where division in implemented in $\mathbb{G}$.*

Note first that correctness holds since party P_1 computes in Step 2 the polynomial evaluation

$$C(t) = \prod_{i=0}^{d} (c_i)^{t^i} = \prod_{i=0}^{d} (g^{q_i}\tilde{r}_i)^{t^i} = \prod_{i=0}^{d} (g^{q_i} g^{r'_i})^{t^i} = g^{Q(t)+R(t)}$$

and then "fixes" its computation by dividing out $Z = g^{R(t)}$. In addition, privacy holds due to the pseudorandomness of PRF that hides the coefficients of $Q(\cdot)$. We prove the following theorem,

Theorem 3.4 *Assume* $\mathrm{PRF} = \langle \mathsf{KeyGen}, \mathsf{PRF}, \mathsf{CFEval} \rangle$ *is an algebraic PRF, then Protocol 2 securely realizes functionality $\mathcal{F}_{\mathrm{OPE}}$ in the presence of semi-honest adversaries in the $\mathcal{F}_{\mathrm{MaskPoly}}$-hybrid model.*

Proof: We prove security for each corruption case separately.

P_0 is corrupted. Let $\mathcal{A}$ be a PPT adversary corrupting party P_0, we design a PPT simulator $\mathcal{S}$ that invokes $\mathcal{A}$ by playing the role of the honest P_1 and generating a view that is indistinguishable from a hybrid view. More concretely,

1. Given input $(1^n, g^{Q(\cdot)}, z)$, $\mathcal{S}$ invokes $\mathcal{A}$ on this input and receives $\mathcal{A}$'s first message, $(\mathbb{G}, p, g)$ and a d-degree polynomial $C(\cdot)$ (where d is also the degree of $Q(\cdot)$).
2. $\mathcal{S}$ emulates the ideal calls of $\mathcal{F}_{\mathrm{MaskPoly}}$, playing the role of the trusted party that receives from $\mathcal{A}$ a PRF key K.
3. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

We prove the following claim,

Claim 3.1 *For any tuple of inputs $(t, g^{Q(\cdot)})$ and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPE}},\mathcal{S}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}} \equiv \{\mathbf{HYBRID}_{\pi^{\mathcal{F}_{\mathrm{MaskPoly}}},\mathcal{A}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}}.$$

Proof: It is trivial to verify that $\mathcal{A}$'s view in the hybrid execution is identically distributed to its view in the ideal execution, since it does not receive any message in the hybrid execution. Now, since correctness holds due to the above argument, then the joint output distribution of both parties is identical in both executions. □

P_1 **is corrupted.** Let $\mathcal{A}$ be a PPT adversary corrupting party P_1, we design a PPT simulator $\mathcal{S}$ that invokes $\mathcal{A}$ by playing the role of the honest P_0 and generating a view that is indistinguishable from a hybrid view. More concretely,

1. Given input $(1^n, t, g^{Q(t)}, z)$, $\mathcal{S}$ invokes $\mathcal{A}$ on $(1^n, t, z)$ and continues as follows.
2. $\mathcal{S}$ invokes first $(\mathbb{G}, p, g) \leftarrow \mathsf{KeyGen}(1^n, \lceil \log d \rceil)$. It then picks a random d-degree polynomial $S(\cdot) = g^{S'(\cdot)}$ in $\mathbb{G}$ (we assume that d is part of $\mathcal{A}$'s auxiliary input, alternatively, d can be part of P_1's output within $\mathcal{F}_{\mathrm{OPE}}$), and sends it to $\mathcal{A}$.
3. $\mathcal{S}$ then emulates the ideal call of $\mathcal{F}_{\mathrm{MaskPoly}}$ by playing the role of the trusted party that receives t from $\mathcal{A}$. $\mathcal{S}$ replies with $g^{S'(t)}/g^{Q(t)}$.
4. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

We prove the following claim,

Claim 3.2 *For any tuple of inputs $(t, g^{Q(\cdot)})$ and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPE}},\mathcal{S}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}} \overset{\mathrm{c}}{\approx} \{\mathbf{HYBRID}_{\pi^{\mathcal{F}_{\mathrm{MaskPoly}}},\mathcal{A}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}}.$$

Proof: We now prove that the adversary's view in both hybrid and ideal executions is computationally indistinguishable by a reduction to the security of PRF. Consider a hybrid execution **Hyb**, for which a simulator $\mathcal{S}_{\mathbf{Hyb}}$ generates $\mathcal{A}$'s view as follows. Assume that there is no trusted party and that $\mathcal{S}_{\mathbf{Hyb}}$ knows the polynomial in the exponent $g^{Q(\cdot)}$, then $\mathcal{S}_{\mathbf{Hyb}}$ first picks a truly random d-degree polynomial $T(\cdot)$ in $\mathbb{Z}_p$ and sends $\mathcal{A}$ the coefficients of the masked polynomial $g^{Q(\cdot)+T(\cdot)}$. Later, it hands $\mathcal{A}$ the value $g^{T(t)}$ within $\mathcal{F}_{\mathrm{MaskPoly}}$. More formally, we mark in bold the underlying differences from the simulation.

1. Given input $(1^n, \mathbf{g}^{\mathbf{Q}(\cdot)}, t, g^{Q(t)}, z)$, $\mathcal{S}_{\mathbf{Hyb}}$ invokes $\mathcal{A}$ on $(1^n, t, z)$ and continues as follows.
2. $\mathcal{S}_{\mathbf{Hyb}}$ invokes first $(\mathbb{G}, p, g) \leftarrow \mathsf{KeyGen}(1^n, \lceil \log d \rceil)$. It then picks a random d-degree polynomial $T(\cdot)$ in $\mathbb{Z}_p$ and sends $\mathcal{A}$ the coefficients of the masked polynomial $\mathbf{g}^{\mathbf{Q}(\cdot)+\mathbf{T}(\cdot)}$.
3. $\mathcal{S}_{\mathbf{Hyb}}$ then emulates the ideal call of $\mathcal{F}_{\mathrm{MaskPoly}}$ by playing the role of the trusted party that receives t from $\mathcal{A}$. $\mathcal{S}_{\mathbf{Hyb}}$ replies with $\mathbf{g}^{\mathbf{T}(\mathbf{t})}$.
4. $\mathcal{S}_{\mathbf{Hyb}}$ outputs whatever $\mathcal{A}$ does.

Our proof follows by proving the following two subclaims.

SubClaim 3.3 *For any tuple of inputs* $(t, g^{Q(\cdot)})$ *and auxiliary input* z*,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\rm OPE},\mathcal{S}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}} \equiv \{\mathbf{Hyb}_{\pi^{\mathcal{F}_{\rm MaskPoly}},\mathcal{A}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}}.$$

Proof: We claim that $\mathcal{A}$'s view in game **Hyb** with $\mathcal{S}'$ is identical to its view in the simulation with $\mathcal{S}$. This is because the random polynomial $S(\cdot) = g^{S'(\cdot)}$ is distributed identically to $g^{T(\cdot)+Q(\cdot)}$ for some polynomial $T(\cdot)$ in $\mathbb{Z}_p$. Therefore, the message $g^{S'(t)}/g^{Q(t)}$ sent by simulator $\mathcal{S}$ within $\mathcal{F}_{\rm MaskPoly}$, is identically distributed to $g^{T(t)+Q(t)}/g^{Q(t)} = g^{T(t)}$. □

Next, we prove that the adversary's view in game **Hyb** and in the hybrid execution **HYBRID** is computationally indistinguishable, relying on the pseudorandomness of PRF.

SubClaim 3.4 *For any tuple of inputs* $(t, g^{Q(\cdot)})$ *and auxiliary input* z*,*

$$\{\mathbf{Hyb}_{\pi^{\mathcal{F}_{\rm MaskPoly}},\mathcal{A}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}} \overset{c}{\approx} \{\mathbf{HYBRID}_{\pi^{\mathcal{F}_{\rm MaskPoly}},\mathcal{A}(z)}(n,(t,g^{Q(\cdot)}))\}_{n\in\mathbb{N}}.$$

Proof: More specifically, assume by contradiction the existence of an adversary $\mathcal{A}$ and a distinguisher $D_{\rm OPE}$ that distinguishes $\mathcal{A}$'s views for infinitely many n's, construct a distinguisher D_F that distinguishes F from a truly random function f that maps $\{0,1\}^m$ to $\mathbb{G}$, for infinitely many n's. Namely, upon given access to an oracle $\mathcal{O}$ that implements either F or f, and auxiliary input $(1^n, g^{Q(\cdot)}, t, z)$, D_F invokes $\mathcal{A}$ on $1^n, t, z$. It then computes the first message of protocol $\Pi_{\rm OPE}$ as follows. It first invokes its oracle on the set $(0,\ldots,d)$; let $f_0,\ldots,f_d$ be the oracle's responses. D_F then sends $\mathcal{A}$ the coefficients $\{g^{q_i}f_i\}_{i=0}^d$, for $g^{Q(\cdot)} = (g^{q_0},\ldots,g^{q_d})$. Next, D_F emulates the trusted party for $\mathcal{F}_{\rm MaskPoly}$, receiving a value t from $\mathcal{A}$ and replying with $\prod_{i=0}^d f_i^{t^i}$. Finally, D_F invokes $D_{\rm OPE}$ on $\mathcal{A}$'s view and outputs whatever $D_{\rm OPE}$ does. Note that if oracle $\mathcal{O}$ implements function F, then $\mathcal{A}$'s view distributes as in the hybrid execution. On the other hand, if $\mathcal{O}$ implements f then $\mathcal{A}$'s view distributes as in the simulation. Therefore, any gap in distinguishing these two views is immediately translated into a gap distinguishing F from f. This concludes the proof. □

□

■

Efficiency. In the first phase P_0 computes $d+1$ modular exponentiations as it can first compute the PRF evaluations in $\mathbb{Z}_p$ (using the PRF key K) and then raise the outcomes to the power of g. Next, P_0 multiplies each PRF evaluation $\mathsf{PRF}_K(i)$ with g^{q_i} (where these computations can be combined into a single exponentiation per index i). Efficiency of the second phase is dominated by the degree of $Q(\cdot)$ and the implementation of functionality $\mathcal{F}_{\rm MaskPoly}$. In Section 3.3 we discuss several ways to realize $\mathcal{F}_{\rm MaskPoly}$. (1) Assuming the strong-DDH assumption, our protocol requires a constant number of modular exponentiations. (2) Assuming the DDH assumption our protocol requires $O(\log d)$ modular exponentiations. Therefore, the overall cost is $2(d+1)+O(1)$ (resp. $O(\log d)$ exponentiations.

Extension for a reactive OPE functionality. In a more realistic setting, we may consider a reactive variant of the OPE functionality where P_0 first commits to its polynomial *once*, similarly to the first message in $\pi_{\rm OPE}$, and then receives multiple requests for polynomial evaluations. These requests can be adaptively sent by either a single or distinct parties. In this case, the amortized work of P_0 would be $O(1)$ (resp. $O(\log d)$) exponentiations for $O(d)$ evaluation requests. Note that such a reactive functionality captures a broader class of applications, e.g., when a single authority communicates with several different entities that wish to securely check membership in the authority's list. This further implies that correctness must be enforced by ensuring that P_0 uses the same PRF key for all of its evaluations; see further discussion in Section 4.

3.2.1 Security in the Presence of Malicious Adversaries

We next prove the security of Protocol 2 in the presence of malicious attacks. We observe that if the protocol that implements $\mathcal{F}_{\mathrm{MaskPoly}}$ is secure in the presence of malicious corruptions then the entire protocol is secure against malicious attacks as well. Intuitively, security against a corrupted P_1 is immediately implied since a corrupted P_1 does not learn anything beyond $g^{R(t')}$, where t' is P_1's input to $\mathcal{F}_{\mathrm{MaskPoly}}$. More concretely, in the security proof the simulator publishes a random polynomial $\widetilde{S}(\cdot) = g^{S(\cdot)}$ first, and then extracts P_1's input t' to π_{MaskPoly}. Finally, the simulator forces P_1's output within π_{MaskPoly} to be $g^{S(t')}/g^{Q(t')}$.

In case P_0 is corrupted we need to demonstrate how to extract the coefficients of $g^{Q(\cdot)}$. This is achieved by the fact that P_0 is committed to the PRF key K within $\mathcal{F}_{\mathrm{MaskPoly}}$. Specifically, extraction is obtained by first calculating the sequence of values $\widetilde{R}(\cdot)$ using K, and then dividing them out (component-wise) from the polynomial $C(\cdot)$ sent by P_0 within the first message. The technical part is to prove that P_1 indeed learns the same value in both ideal and real executions. Intuitively, by the correctness of the ideal execution for functionality $\mathcal{F}_{\mathrm{MaskPoly}}$ we are ensured that P_1 learns $g^{R(t)}$ such that $\widetilde{R}(x) = g^{R(x)} = \tilde{r}_0 \cdot \tilde{r}_1^x \cdot \ldots \cdot \tilde{r}_d^{x^d}$ where $\tilde{r}_i = \mathsf{PRF}_K(i)$ for all i and K is the PRF key as above. This implies that P_1 outputs $C(t)/g^{R(t)}$, which corresponds to a polynomial evaluation with the same coefficients as defined above by the simulator.

To conclude, in order to obtain malicious security the only modification we need to consider with respect to π_{OPE} is to employ a maliciously secure implementation of functionality $\mathcal{F}_{\mathrm{MaskPoly}}$. In the hybrid setting this does not make a difference for the protocol description. In Section 3.3 we discuss secure implementations of functionality $\mathcal{F}_{\mathrm{MaskPoly}}$. The proof for the following theorem formalizes the intuition above.

Theorem 3.5 *Assume* $\mathrm{PRF} = \langle \mathsf{KeyGen}, \mathsf{PRF}, \mathsf{CFEval} \rangle$ *is an algebraic PRF, then Protocol 2 securely realizes functionality* $\mathcal{F}_{\mathrm{OPE}}$ *in the presence of malicious adversaries in the* $\mathcal{F}_{\mathrm{MaskPoly}}$*-hybrid model.*

Proof: We prove security for each corruption case separately.

P_0 **is corrupted.** Let $\mathcal{A}$ be a PPT adversary corrupting party P_0, we design a PPT simulator $\mathcal{S}$ that extracts the coefficients of $\mathcal{A}$'s input polynomial in the exponent of some generator g, and sends them to the trusted party. We prove that the joint output distribution in both ideal and hybrid executions is computationally indistinguishable.

1. Given input $(1^n, g^{Q(\cdot)}, z)$, $\mathcal{S}$ invokes $\mathcal{A}$ on this input and receives $\mathcal{A}$'s first message, $(\mathbb{G}, p, g)$ and a d-degree polynomial $C(\cdot) = (c_0, \ldots, c_d)$.

2. $\mathcal{S}$ emulates the ideal calls of $\mathcal{F}_{\mathrm{MaskPoly}}$, playing the role of the trusted party that receives from $\mathcal{A}$ a PRF key K.

3. $\mathcal{S}$ defines polynomial $g^{Q'(\cdot)}$ with the following set of coefficients. Namely, $\mathcal{S}$ fixes $g^{q'_i} = c_i/\tilde{r}_i$ for all $0 \leq i \leq d$ where $\tilde{r}_i = \mathsf{PRF}_K(i)$. $\mathcal{S}$ sends the trusted party the set $\{g^{q'_i}\}_i$.

4. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

Note that $\mathcal{A}$ receives no messages in the hybrid version of our protocol. Therefore security is implied by correctness of the closed form efficiency property that implies that the coefficients defined by the simulator implies a polynomial evaluation that is consistent with the real result.

P_1 **is corrupted.** Let $\mathcal{A}$ be a PPT adversary corrupting party P_1, we design a PPT simulator $\mathcal{S}$ that generates the view of $\mathcal{A}$ as in the semi-honest case with the following difference. $\mathcal{S}$ invokes $\mathcal{A}(1^n, t, z)$ and sends it $(\mathbb{G}, p, g)$ and a random polynomial $\widetilde{S}(\cdot) = g^{S(\cdot)}$. Upon receiving the adversary's input t' to $\mathcal{F}_{\mathrm{MaskPoly}}$,

$\mathcal{S}$ forwards it to the trusted party for $\mathcal{F}_{\rm OPE}$. Let Z denotes the output returned by the trusted party, then $\mathcal{S}$ completes the simulation, forcing the output of $\mathcal{F}_{\rm MaskPoly}$ to be $g^{S(t')}/Z$. We note that security follows exactly as in the semi-honest proof since $\mathcal{A}$ sees the same message distribution. ■

3.3 Malicious Security of $\pi_{\rm MaskPoly}$

For completeness, we describe a concrete protocol that implements functionality $\mathcal{F}_{\rm MaskPoly} : (K,t) \mapsto (-, g^{R(t)})$, used as a subprotocol within our main protocol $\pi_{\rm OPE}$ for oblivious polynomial evaluation from Section 3.2. This computation corresponds to the polynomial evaluation $\widetilde{R}(x) = \mathsf{PRF}_K(0)\cdot\mathsf{PRF}_K(1)^x\cdot\ldots\cdot\mathsf{PRF}_K(d)^{x^d}$ with respect to function PRF. In what follows, we discuss a detailed secure implementation for PRF_1 that is described in Section 3.1.1 and then briefly discuss how to implement function PRF_2, formally described in Section 3.1.2, using similar ideas. Alternative suggestions based on general-purpose protocols or somewhat homomorphic encryption scheme that supports a limited number of multiplications, as we require for our PRFs.

We recall that when implementing functionality $\mathcal{F}_{\rm MaskPoly}$ relative to PRF_1 the parties compute the value

$$g^{\sum_{i=0}^{d} k_0 k_1^i x^i} = \frac{k_0((k_1 t)^{d+1}-1)}{k_1 t - 1} = \frac{k_0(k_1 t)^{d+1}}{k_1 t - 1} - \frac{k_0}{k_1 t - 1} = \frac{k_0 k_1^{d+1} t^{d+1}}{k_1 t - 1} - \frac{k_0}{k_1 t - 1},$$

so that P_0 enters a PRF key $K = (k_0, k_1)$ and learns nothing and P_1 enters $x = t$ and learns this outcome. This simple computation requires a constant number of exponentiations and can be easily implemented securely. Achieving malicious security requires to ensure correctness of computations which we obtain using simple zero-knowledge proofs of knowledge. Loosely speaking, the parties first generate a joint public key for the additive El Gamal PKE such that no party knows the secret key (this protocol is discussed in Section 2.4.1). Next, each party commits to its input and the parties jointly compute $k_1 t$. In order to compute the inverse of $k_1 t - 1$ the parties additionally use an additively homomorphic encryption scheme (such as Paillier [Pai99]). Formally, our protocol uses the following tools:

1. Distributed additive El Gamal; see Section 2.4 for the details of these schemes. We denote this scheme by $\Pi = (\pi_{\rm KeyGen}, \mathsf{Enc}, \pi_{\rm DEC})$.
2. Additively homomorphic encryption scheme $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$.
3. Zero-knowledge proofs of knowledge: $\pi_{\rm DL}$ for proving a discrete logarithm and $\pi_{\rm EQ}, \pi_{\rm Eq-Exp}$ for proving consistency of exponents, which are formally stated in Section 2.5. For simplicity, we implicitly assume that the parties continue to the next step only if all the proofs were verified correctly thus far.

Finally, we implicity assume that a party rerandomizes its homomorphic computations on the ciphertexts. Such that rerandomization is carried out by multiplying the outcome with a random encryption of zero. We now describe our protocol is details.

Protocol 3 (Protocol $\pi_{\rm MaskPoly}$ with malicious security.)

- **Input:** *Party P_0 is given a PRF key $K = (k_0, k_1)$. Party P_1 is given an element t. Both parties are given a security parameter 1^n, a polynomial degree d and $(\mathbb{G}, p, g)$ for a group description $\mathbb{G}$ of prime order p and a generator g.*
- **The protocol:**
  1. **Distributed key generation.** *P_0 and P_1 run the protocol $\pi_{\rm KeyGen}(1^n, 1^n)$ in order to generate an additive El Gamal public key* $\mathrm{PK} = \langle \mathbb{G}, p, g, h\rangle$ *for which the corresponding shares of the secret key* SK *are* $(\mathrm{SK}_0, \mathrm{SK}_1)$.

Next, P_0 sends P_1 encryptions of k_0 and k_1, respectively denoted by c_{k_0} and c_{k_1}, and proves their knowledge using $\pi_{\rm 2DL}$. Namely, given an El Gamal ciphertext $\langle c_1, c_2\rangle = \langle g^r, h^r g^m\rangle$, the proven statement is c_2 for which P_0 proves that it knows r and m.

2. **Computing encryption of (masked) $\mathbf{k_1 t - 1}$.** *P_0 runs $(\mathrm{PK}', \mathrm{SK}') \leftarrow \mathsf{Gen}'(1^n)$ and then sends P_1 the key PK' and an encryption of k_1, denoted by $c'_{k_1} = \mathsf{Enc}'_{pk'}(k_1)$, and proves consistency with c_{k_1} using $\pi_{\rm Eq-Exp}$. P_1 responds with $c' = \mathsf{Enc}'_{\mathrm{PK}'}\big((k_1 t - 1)\cdot r\big)$ and $c_r = \mathsf{Enc}_{\mathrm{PK}}(r)$ for some random mask r.*[4]
3. **Computing encryption of $\mathbf{(k_1 t - 1)^{-1}}$.** *Upon receiving ciphertext c' from P_1, P_0 decrypts it and re-encrypts it, as well as the inverse of this plaintext modulo p, under PK, creating ciphertexts $c, c_{\rm INV}$ and proving consistency between c' and c using $\pi_{\rm Eq-Exp}$, and between c and $(c_{\rm INV}, \mathsf{Enc}_{\mathrm{PK}}(g^1))$ using $\pi_{\rm MULT}$, where the latter ciphertext is generated by P_0 and proven correct using $\pi_{\rm DDH}$.*

   As a response, P_1 unmasks r from the plaintext within ciphertext c by raising it to the power of r, creating ciphertext $c_{r^{-1}}$ and proving consistency with c_r using $\pi_{\rm EQ}$.
4. **Computing encryptions of $\mathbf{k_1^{d+1}}$ and $\mathbf{t^{d+1}}$.** *P_0 computes an encryption of k_1^{d+1}, denoted by $c_{k_1^{d+1}}$, and proves consistency between g^{d+1} and $c_{k_1^{d+1}}$ using $\pi_{\rm EQ}$. Similarly, P_0 computes the encryption of t^{d+1}, denoted by $c_{t^{d+1}}$, and proves correctness (where an encryption of t can be recovered from the encryption of $k_1 t - 1$, denoted above by $c_{r^{-1}}$).*
5. **Computing encryptions of $\mathbf{k_0(k_1 t - 1)^{-1}}$ and $\mathbf{k_0 k_1^{d+1}(k_1 t - 1)^{-1}}$.** *Given ciphertexts $c_{\rm INV}$, $c_{k_1^{d+1}}$ and c_{k_0}, P_0 computes the encryptions of $k_0(k_1 t - 1)^{-1}$ and $k_0 k_1^{d+1}(k_1 t - 1)^{-1}$ and proves consistency relative to $c_{\rm INV}$, $c_{k_1^{d+1}}$ and c_{k_0} using $\pi_{\rm EQ}$ (where the proof of the later computation involves running $\pi_{\rm EQ}$ twice). Let c_0 and c'_0 denote the respective outcomes.*
6. **Computing encryption of $\mathbf{k_0 k_1^{d+1} t^{d+1}(k_1 t - 1)^{-1}}$.** *Given ciphertexts $c_{t^{d+1}}$ and c'_0, P_1 computes the encryption of $k_0 k_1^{d+1} t^{d+1}(k_1 t - 1)^{-1}$ and proves consistency using $\pi_{\rm EQ}$. Let c_1 denote the respective outcome.*
7. **Outcome.** *Finally, the parties decrypt c_1/c_0 for P_1 by running π_{Dec}, that outputs the result.*

Theorem 3.6 *Assume $\Pi = (\pi_{\rm KeyGen}, \mathsf{Enc}, \pi_{\rm DEC})$, $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ are as above, then Protocol 3 securely realizes functionality $\mathcal{F}_{\rm MaskPoly}$ with respect to PRF_1 in the presence of malicious adversaries in the $\{\mathcal{F}_{\rm 2DL}, \mathcal{F}_{\rm Eq-Exp}, \mathcal{F}_{\rm MULT}, \pi_{\rm DDH}, \mathcal{F}_{\rm EQ}\}$-hybrid model.*

Proof Sketch: Informally, simulation follows since the parties are forced to prove their computations using ZK proofs which guarantees correctness. More concretely, in case P_0 is corrupted, the simulator extracts k_0 and k_1 from the ZK proof $\pi_{\rm 2DL}$ in Step 1, and sends these values to the trusted party. It further plays the role of the honest P_1 using an arbitrary input and aborts if one of the proofs is not verified correctly. Security is reduced to the IND-CPA security of the distributed El Gamal PKE since P_0 does not learn any output. Moreover, the plaintext decrypted within the Paillier ciphertext is masked using a random string r.

On the other hand, in case P_1 is corrupted the simulation follows by having the simulator use an arbitrary PRF key and extract t from the proof that verifies the computation of $k_1 t$ in Step 3, except that the simulator also cheats in the mutual decryption in the final step and forces the result to be the output it received from the trusted party. Security in this case follows due to the IND-CPA security of the distributed El Gamal PKE and the security of protocol $\pi_{\rm DEC}$. Finally, we note that the complexity of our protocol is constant. ■

The implementation of PRF PRF_2 follows similarly. Namely, recall that the parties compute the value $g^{k_0(1+k_1,x)(1+k_2x^2)\ldots(1+k_m x^{2^m})}$ which can be carried out in $O(m)$ overhead as follows. First, P_0 commits to its key $(k_0, k_1, \ldots, k_m)$, whereas P_1 commits to the elements $(x, x^2, \ldots, x^{2^m})$ together with a ZK proof of consistency $\pi_{\rm MULT}$ for any consecutive pair of elements, proving that the ith encryption encrypts the

[4]We assume that the message space $\mathbb{G}'$ of Π' is large enough so that a plaintext of Π is not wrapped within $\mathbb{G}'$ and that r is large enough to mask $k_1 t - 1$.

products of the $i-1$ plaintext with itself. Next, given the product $\tilde{g} = g^{k_0(1+k_1,x)(1+k_2x^2)\dots(1+k'_m x^{2^{m'}})}$ for some integer $m' < m$, the parties mutually compute

$$\tilde{g} \cdot \tilde{g}^{k_{m'+1}x^{2^{(m'+1)}}} = \tilde{g}^{(1+k_{m'+1}x^{2^{(m'+1)}})} = g^{k_0(1+k_1,x)(1+k_2x^2)\dots(1+k_{m'+1}x^{2^{(m'+1)}})}$$

where $\hat{g} = \tilde{g}^{k_{m'+1}}$ is carried out by P_0 and proven correct with respect the commitment of $g^{k_{m'+1}}$. This computation is followed by P_1 computing $\hat{g}^{x^{2^{(m'+1)}}}$ which is also verified against the commitment of $g^{x^{2^{(m'+1)}}}$ where the commitment is realized using El Gamal. See the ZK proof π_{EQ} for more details.

4 Secure Set-Intersection

One important application that benefits from our OPE construction is the set-intersection functionality which is defined by having each party's input consists of a *set* of elements from domain $\{0,1\}^t$. Formally:

Definition 4.1 *Let X and Y be subsets of a predetermined arbitrary domain $\{0,1\}^t$ and m_X and m_Y the respective upper bounds on the sizes of X and Y.*[5] *Then functionality $\mathcal{F}_\cap$ is defined by:*

$$(X,Y) \mapsto (m_Y, (X \cap Y, m_X)).$$

We modify protocol π_{OPE} from Section 3.2.1 as follows. First, P_0 prepares a polynomial $Q(\cdot)$ with coefficients in $\mathbb{Z}_p$ and the set of roots X. It then masks $Q(\cdot)$ as in Protocol 2 using the sequence of pseudorandom elements $\widetilde{R}(\cdot)$. The parties then interact with a trusted party that computes functionality $\mathcal{F}_{\mathrm{EqMask}}$, which is a variation of functionality $\mathcal{F}_{\mathrm{MaskPoly}}$ defined as follows:

Definition 4.2 *Let X and Y be subsets of a predetermined arbitrary domain $\{0,1\}^t$ and m_X and m_Y the respective upper bounds on the sizes of X and Y. Then functionality $\mathcal{F}_{\mathrm{EqMask}}$ is defined by:*

$$(K, \{(y, T_y)\}_{y \in Y}) \mapsto (-, \{b_i\}_i)$$

where $b_i = 1$ only if $T_y = g^{R(y)}$ and 0 otherwise, $g^{R(\cdot)} = \left(g^{r_0}, \dots, g^{r_{m_X}}\right)$ and $g^{r_i} = \mathsf{PRF}_K(i)$ for all i. Stated differently, $b_i = 1$ if and only if $Q(y) = 0$ (or $y \in X \cap Y$) with overwhelming probability.

Finally, P_1 outputs the set of elements $Z \subseteq Y$ for which $b_i = 1$.

4.1 Realizing $\mathcal{F}_{\mathrm{EqMask}}$

The realization of $\mathcal{F}_{\mathrm{MaskPoly}}$ from Section 3.3 for PRF PRF_1 can be modified to realize $\mathcal{F}_{\mathrm{EqMask}}$ by having P_0 run its proofs with respect to a single set of ciphertexts encrypting its PRF key. More specifically, the parties generate a single set of keys for the two encryption schemes. Following that, P_0 encrypts k_0 and k_1 only once. The rest of the computation follows for every $y \in Y$ in parallel, where all the ZK proofs are parallely executed (relying on the fact that they were all constructed as Σ-protocols). Finally, decryption is carried out using protocol π_{DEC_0} discussed in Section 2.4.1.

Protocol 4 (Protocol π_{EqMask} with malicious security.)

- **Input:** *Party P_0 is given a PRF key $K = (k_0, k_1)$. Party P_1 is given $\{(y, T_y)\}_{y \in Y}$. Both parties are given a security parameter 1^n and $(\mathbb{G}, p, g)$ for a group description $\mathbb{G}$ of prime order p and a generator g.*

[5]In order to deal with a proof technicality, where a corrupted party inputs less elements than its set size, prior constructions assume a super polynomial lower bound on the input domain sizes. Since we do not wish to restrict the input domains, we assume that the set sizes are not strict and may denote some upper bound on the actual numbers of elements.

- **The protocol:**

  1. **Distributed key generation.** *P_0 and P_1 run the protocol $\pi_{\rm KeyGen}(1^n, 1^n)$ in order to generate an additive El Gamal public key $\mathrm{PK} = \langle \mathbb{G}, p, g, h\rangle$ for which the corresponding shares of the secret key SK are $(\mathrm{SK}_0, \mathrm{SK}_1)$.*

     Next, P_0 sends P_1 encryptions of k_0 and k_1, respectively denoted by c_{k_0} and c_{k_1}, and proves their knowledge using $\pi_{\rm 2DL}$. Namely, given an El Gamal ciphertext $\langle c_1, c_2\rangle = \langle g^r, h^r g^m\rangle$, the proven statement is c_2 for which P_0 proves that it knows r and m.
  2. **Generating additively homomorphic keys.** *P_0 runs $(\mathrm{PK}', \mathrm{SK}') \leftarrow \mathsf{Gen}'(1^n)$ and sends PK' to P_1, as well as an encryption of k_1, denoted by $c'_{k_1} = \mathsf{Enc}'_{pk'}(k_1)$. P_0 proves consistency with c_{k_1} using $\pi_{\rm Eq-Exp}$.*

     Next, the following steps are performed for any element y from the set Y in parallel.
  3. **Computing encryption of (masked) $\mathbf{k_1 y - 1}$.** *P_1 responds with $c' = \mathsf{Enc}'_{\mathrm{PK}'}\big((k_1 y - 1)\cdot r\big)$ and $c_r = \mathsf{Enc}_{\mathrm{PK}}(r)$ for some random mask r.*[6]
  4. **Computing encryption of $\mathbf{(k_1 y - 1)^{-1}}$.** *Upon receiving ciphertext c' from P_1, P_0 decrypts it and re-encrypts it, as well as the inverse of this plaintext modulo p, under PK, creating ciphertexts $c, c_{\rm INV}$ and proving consistency between c' and c using $\pi_{\rm Eq-Exp}$, and between c and $(c_{\rm INV}, \mathsf{Enc}_{\mathrm{PK}}(g^1))$ using $\pi_{\rm MULT}$, where the latter ciphertext is generated by P_0 and proven correct using $\pi_{\rm DDH}$.*

     As a response, P_1 unmasks r from the plaintext within ciphertext c by raising it to the power of $r \bmod p$, creating ciphertext $c_{r^{-1}}$ and proving consistency with c_r using $\pi_{\rm EQ}$.
  5. **Computing encryptions of $\mathbf{k_1^{d+1}}$ and $\mathbf{y^{d+1}}$.** *P_0 computes an encryption of k_1^{d+1}, denoted by $c_{k_1^{d+1}}$, and proves consistency between g^{d+1} and $c_{k_1^{d+1}}$ using $\pi_{\rm EQ}$. Similarly, P_0 computes the encryption of y^{d+1}, denoted by $c_{y^{d+1}}$, and proves correctness (where an encryption of y can be recovered from the encryption of $k_1 y - 1$, denoted above by $c_{r^{-1}}$).*
  6. **Computing encryptions of $\mathbf{k_0(k_1 y - 1)^{-1}}$ and $\mathbf{k_0 k_1^{d+1}(k_1 y - 1)^{-1}}$.** *Given ciphertexts $c_{\rm INV}$, $c_{k_1^{d+1}}$ and c_{k_0}, P_0 computes the encryptions of $k_0(k_1 y - 1)^{-1}$ and $k_0 k_1^{d+1}(k_1 y - 1)^{-1}$ and proves consistency relative to $c_{\rm INV}$, $c_{k_1^{d+1}}$ and c_{k_0} using $\pi_{\rm EQ}$ (where the proof of the later computation involves running $\pi_{\rm EQ}$ twice). Let c_0 and c'_0 denote the respective outcomes.*
  7. **Computing encryption of $\mathbf{k_0 k_1^{d+1} y^{d+1}(k_1 y - 1)^{-1}}$.** *Given ciphertexts $c_{y^{d+1}}$ and c'_0, P_1 computes the encryption of $k_0 k_1^{d+1} y^{d+1}(k_1 y - 1)^{-1}$ and proves consistency using $\pi_{\rm EQ}$. Let c_1 denote the respective outcome.*
  8. **Outcome.** *Finally, the parties run protocol $\pi_{{\rm DEC}_0}$ on $(c_1/c_0)/T_y$ for P_1, that outputs y if the decryption result is zero.*

Theorem 4.3 *Assume $\Pi = (\pi_{\rm KeyGen}, \mathsf{Enc}, \pi_{\rm DEC})$, $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ are as in Section 3.3, then Protocol 4 securely realizes functionality $\mathcal{F}_{\rm EqMask}$ with respect to PRF_1 in the presence of malicious adversaries in the $\{\mathcal{F}_{\rm 2DL}, \mathcal{F}_{\rm Eq-Exp}, \mathcal{F}_{\rm MULT}, \pi_{\rm DDH}, \mathcal{F}_{\rm EQ}\}$-hybrid model.*

Intuitively speaking, security follows similarly to the proof of Protocol 3 due to forcing the parties to enclose a zero-knowledge proof for each step of computation. In the proof, the simulator extracts P_0's input from the first step, whereas input extraction of P_1's input is concluded from the computations made in the third step under PK' and the masking r determined by P_0 in the forth step. As in Section 3.3, this protocol can be easily extended to realize $\mathcal{F}_{\rm EqMask}$ relative to PRF_2.

4.2 The Complete Protocol

Having described a protocol for realizing $_{\rm EqMask}$, we are ready to describe the complete protocol for set-intersection. Intuitively speaking, in order to enable the extraction of the set X by the simulator we add

[6]See Footnote 4.

zero-knowledge proofs of knowledge for the relation $\mathcal{R}_{\mathrm{DL}}$, formally defined in Section 2.5. This technicality arises because P_0 sends elements in $\mathbb{G}$ yet the polynomial $Q(\cdot) + R(\cdot)$ is evaluated *in the exponent*, implying that X and Y must be sampled from $\mathbb{Z}_p$ as well. In addition, extraction of Y follows via π_{EqMask}. Formally, we define our set-intersection protocol as follows,

Protocol 5 (Protocol $\pi_\cap$ with malicious security.)

- **Input:** *Party P_0 is given a set X of size m_X. Party P_1 is given a set Y of size m_Y. Both parties are given a security parameter 1^n.*
- **The protocol:**
  1. **Masking the input polynomial.** *P_0 defines an d-degree polynomial $Q(\cdot) = (q_0, \ldots, q_d)$ with coefficients in $\mathbb{Z}_p$ and the set of roots X, for $d = m_X$. It then invokes $(K, param) \leftarrow \mathsf{KeyGen}(1^n, d)$ where $param$ includes a group description $\mathbb{G}$ of prime order p and a generator g, and defines a new d-degree polynomial $\widetilde{R}(\cdot) = (\tilde{r}_0, \ldots, \tilde{r}_d)$ over $\mathbb{G}$, where r_i is defined by $\mathsf{PRF}_K(i)$ for all i.*

     P_0 sends P_1 $param$ and the masked polynomial $C(\cdot) = \left(g^{q_0}\tilde{r}_0, \ldots, \ldots, g^{q_d}\tilde{r}_d\right)$, where multiplication is implemented in $\mathbb{G}$. P_0 further proves the knowledge of the discrete logarithm of $c_i = g^{q_i}\tilde{r}_i$ for all i with respect to a generator g, by invoking π_{DL} on input $\{((g, c_i), \log_g c_i)\}_{i\in[0,d]}$.[7]
  2. **Unmasking the result.** *Upon receiving the masked polynomial $C(\cdot) = (c_0, \ldots, c_d)$ and upon accepting the proofs π_{DL} for all i, party P_1 computes the polynomial evaluation $C(y) = \prod_{i=0}^{d}(c_i)^{y^i}$ for all $y \in Y$ (picked in a random order). I.e., $C(\cdot)$ is evaluated* in the exponent.

     Next, the parties invoke an ideal execution of $\mathcal{F}_{\mathrm{EqMask}}$, where the input of P_0 is K and the input of P_1 is the set $\{(y, C(y))\}_{y\in Y}$. P_1 outputs y if and only if the output from $\mathcal{F}_{\mathrm{EqMask}}$ on $(y, C(y))$ is 1.

Correctness follows easily since P_1 outputs only elements in Y that zeros polynomial $Q(\cdot)$, whose its roots are the set X. Next, we prove the following theorem.

Theorem 4.4 *Assume* $\mathrm{PRF} = \langle\mathsf{KeyGen}, F, \mathsf{CFEval}\rangle$ *is an algebraic PRF, then Protocol 5 securely realizes functionality $\mathcal{F}_\cap$ in the presence of malicious adversaries in the $\{\mathcal{F}_{\mathrm{DL}}, \mathcal{F}_{\mathrm{EqMask}}\}$-hybrid model.*

The proof follows similarly to the proof of Protocol 2. Formally,

Proof: We prove security for each corruption case separately.

P_0 is corrupted. Let $\mathcal{A}$ be a PPT adversary corrupting party P_0, we design a PPT simulator $\mathcal{S}$ that simulates the view $\mathcal{A}$, playing the role of the honest P_1 while extracting $\mathcal{A}$'s input set X, details follow.

1. Given input $(1^n, X, z)$, $\mathcal{S}$ invokes $\mathcal{A}$ on this input and receives $\mathcal{A}$'s first message, $(\mathbb{G}, p, g)$ and a d-degree polynomial $C(\cdot) = (c_0, \ldots, c_d)$.
2. $\mathcal{S}$ emulates the ideal calls of $\mathcal{F}_{\mathrm{DL}}$ by playing the role of the trusted party that receives from $\mathcal{A}$ tuples $\{((g, c_i), c'_i)\}_{i\in[0,d]}$ and records these values. $\mathcal{S}$ verifies whether $c_i = g^{c'_i}$ for all i and records 1 only if these conditions are met, and 0 otherwise. In case $\mathcal{S}$ records 0 it aborts and outputs whatever $\mathcal{A}$ does.
3. $\mathcal{S}$ defines the input set X' as follows. For every i let $\tilde{r}_i = \mathsf{PRF}_K(i)$ and $r_i = \log_g \tilde{r}_i$ and let $q'_i = c'_i - r_i$.[8] $\mathcal{S}$ fixes polynomial $Q'(\cdot) = (q'_0, \ldots, q'_d)$ and defines X' to be the set of roots of $Q'(\cdot)$. $\mathcal{S}$ computes X' by factoring $Q'(\cdot)$ over $\mathbb{Z}_p$ and sends the set X' to the trusted party, receiving back m_Y.

[7]We implicitly assume that P_0 knows the discrete logarithms of the r_i's by its knowledge of K. This is the case for all PRF implementations presented in [BGV11].

[8]See Footnote 7.

4. $\mathcal{S}$ emulates the ideal call of $\mathcal{F}_{\text{MaskPoly}}$ by playing the role of the trusted party that receives from $\mathcal{A}$ a PRF key K.

5. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

We next prove the following claim.

Claim 4.1 *For any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_\cap,\mathcal{S}(z)}(n,(X,Y))\}_{n\in\mathbb{N}} \equiv \{\mathbf{HYBRID}_{\pi_\cap^{\mathcal{F}_{\text{DL}},\mathcal{F}_{\text{EqMask}}},\mathcal{A}(z)}(n,(X,Y))\}_{n\in\mathbb{N}}.$$

Proof: Note that the adversary's view is identical to its view in the hybrid execution since it does not get any output from the internal ideal calls as well as from $\mathcal{F}_\cap$. We now claim that P_1's output is identical with overwhelming probability in both executions due to the following. In the hybrid execution the correctness of the ideal call for $\mathcal{F}_{\text{EqMask}}$ ensures that P_1 obtains the correct equality bit for every $y \in Y$. Namely, if $C(y) \neq \widetilde{R}(y)$ then the honest P_1 obtains 0 from $\mathcal{F}_{\text{EqMask}}$ and does not output y. On the other hand, if $C(y) = \widetilde{R}(y)$ then P_0 receives 1 and returns y. Stating differently, P_1 returns $y \in Y$ only if $C(y)/\widetilde{R}(y) = 1$ where division is computed component-wise. Next, in the simulation $\mathcal{S}$ defines the input set X' of the adversary as the set of roots with respect to the unmasked polynomial $C(\cdot)/\widetilde{R}(\cdot)$ (computed component-wise), where the masking is defined by the PRF key K input by the adversary to $\mathcal{F}_{\text{EqMask}}$. Therefore the intersection is computed with respect to the same set X'. □

P_1 **is corrupted.** Let $\mathcal{A}$ be a PPT adversary corrupting party P_1, we design a PPT simulator $\mathcal{S}$ that generates the view of $\mathcal{A}$ as follows. $\mathcal{S}$ first sends a random polynomial $\widetilde{S}(\cdot)$. Next, upon receiving the adversary's set of elements Y' to $\mathcal{F}_{\text{MaskPoly}}$, $\mathcal{S}$ forwards it to the trusted party for $\mathcal{F}_\cap$. Let Z' denotes the output returned by the trusted party, then $\mathcal{S}$ completes the simulation by forcing the output of $\mathcal{A}$ within $\mathcal{F}_{\text{EqMask}}$ to be consistent with the set Z. More formally,

1. Given input $(1^n, Y, z)$, $\mathcal{S}$ invokes $\mathcal{A}$ on this input and sends it $(\mathbb{G}, p, g)$.

2. $\mathcal{S}$ picks a random d-degree polynomial $\widetilde{S}(\cdot) = (\tilde{s}_0, \ldots, \tilde{s}_d) = (g^{s_0}, \ldots, g^{s_d}) = g^{S(\cdot)}$ with coefficients in $\mathbb{G}$ and sends it to $\mathcal{A}$. (We assume that the simulator knows m_X as part of its auxiliary information. This can also be assured by modifying the definition of the functionality, given m_X to P_1 as part of its input).

3. $\mathcal{S}$ emulates the ideal calls of $\mathcal{F}_{\text{DL}}$ by playing the role of the trusted party that receives from $\mathcal{A}$ tuples $\{(g, \tilde{s}_i)\}_{i\in[0,d]}$ and sends $\mathcal{A}$ the value 1 for all i (denoting accept calls).

4. $\mathcal{S}$ then emulates the ideal call of $\mathcal{F}_{\text{EqMask}}$ by playing the role of the trusted party that receives from $\mathcal{A}$ the set $\{(y'_j, T_{y'_j})\}_{j\in[m_Y]}$. $\mathcal{S}$ sends the set $Y' = \{y'_j\}_{j\in[m_Y]}$ to the trusted party, receiving back the intersection $Z = X \cap Y'$.

   For all $y'_j \in Z$, $\mathcal{S}$ emulates the ideal response of $\mathcal{F}_{\text{EqMask}}$ as follows. If $T_{y'_j} = g^{S(y'_j)}$ then $\mathcal{S}$ sends $\mathcal{A}$ the value 1. Otherwise it sends 0. For all $y'_j \notin Z$, $\mathcal{S}$ always replies with 0.

5. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

We next prove the following claim.

Claim 4.2 *For any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_\cap,\mathcal{S}(z)}(n,(X,Y))\}_{n\in\mathbb{N}} \overset{c}{\approx} \{\mathbf{HYBRID}_{\pi_\cap^{\mathcal{F}_{\text{DL}},\mathcal{F}_{\text{EqMask}}},\mathcal{A}(z)}(n,(X,Y))\}_{n\in\mathbb{N}}.$$

Proof: Note that the differences between the hybrid and simulated executions are as follows. First, $\mathcal{S}$ sends in the simulation a random polynomial instead of a real masked polynomial. In addition, $\mathcal{S}$ fixes the output of $\mathcal{F}_{\text{EqMask}}$ based on the correctness of $\mathcal{A}$'s computations which deviates from the way this functionality is defined. Our proof follows by defining a new hybrid game and proving indistinguishability from both the simulated and the hybrid executions.

Hybrid **Hyb**. Consider a hybrid game **Hyb** where there is no trusted party and the simulator $\mathcal{S}_{\textbf{Hyb}}$ uses the real input X of P_0 to define the polynomial $Q(\cdot)$, but decides on the output of $\mathcal{F}_{\text{EqMask}}$ according to the strategy specified in the simulation. Namely for every pair $(y'_j, T_{y'_j})$, $\mathcal{S}_{\textbf{Hyb}}$ verifies first whether $T_{y'_j} = \tilde{R}(y'_j)$ and returns 1 if equality holds (where $\tilde{R}(\cdot)$ is as defined in the protocol). We next prove that the views induced in **Hyb** and in the simulation are computationally indistinguishable due to the pseudorandomness of F. This argument will be similar to the argument presented in the proof of Protocol 2. More formally, we mark in bold the underlying differences from the simulation.

1. Given input $(1^n, \mathbf{X}, Y, z)$, $\mathcal{S}_{\textbf{Hyb}}$ invokes $\mathcal{A}$ on this input and sends it $(\mathbb{G}, p, g)$.

2. $\mathcal{S}_{\textbf{Hyb}}$ defines an d-degree polynomial $Q(\cdot) = (q_0, \ldots, q_d)$ with coefficients in $\mathbb{Z}_p$ and the set of roots X, for $d = m_X$. It then invokes $(K, param) \leftarrow \mathsf{KeyGen}(1^n, d)$ where $param$ includes a group description $\mathbb{G}$ of prime order p and a generator g, and defines a **new d-degree polynomial** $\widetilde{\mathbf{R}}(\cdot) = (\tilde{\mathbf{r}}_\mathbf{0}, \ldots, \tilde{\mathbf{r}}_\mathbf{d})$ **over** $\mathbb{G}$**, where** $\mathbf{r}_i$ **is defined by** $\mathsf{PRF}_\mathbf{K}(\mathbf{i})$ **for all i**.

   $\mathcal{S}_{\textbf{Hyb}}$ sends $\mathcal{A}$ $param$ and the masked polynomial $C(\cdot) = \left(g^{q_0}\tilde{r}_0, \ldots, \ldots, g^{q_d}\tilde{r}_d\right)$, where multiplication is implemented in $\mathbb{G}$.

3. $\mathcal{S}_{\textbf{Hyb}}$ emulates the ideal calls of $\mathcal{F}_{\text{DL}}$ by playing the role of the trusted party that receives from $\mathcal{A}$ tuples $\{(g, \tilde{s}_i)\}_{i\in[0,d]}$ and sends $\mathcal{A}$ the value 1 for all i (denoting accept calls).

4. $\mathcal{S}_{\textbf{Hyb}}$ then emulates the ideal call of $\mathcal{F}_{\text{EqMask}}$ by playing the role of the trusted party that receives from $\mathcal{A}$ the set $\{(y'_j, T_{y'_j})\}_{j\in[m_Y]}$. $\mathcal{S}_{\textbf{Hyb}}$ sends the set $Y' = \{y'_j\}_{j\in[m_Y]}$ to the trusted party, receiving back the intersection $Z = X \cap Y'$.

   For all $y'_j \in Z$, $\mathcal{S}_{\textbf{Hyb}}$ emulates the ideal response of $\mathcal{F}_{\text{EqMask}}$ as follows. If $T_{y'_j} = g^{S(y'_j)}$ then $\mathcal{S}_{\textbf{Hyb}}$ sends $\mathcal{A}$ the value 1. Otherwise it sends 0. For all $y'_j \notin Z$, $\mathcal{S}_{\textbf{Hyb}}$ always replies with 0.

Next, we prove that the distributions induced by the views of the ideal execution and game **Hyb** are statistically close.

SubClaim 4.3 *For any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\textbf{IDEAL}_{\mathcal{F}_\cap, \mathcal{S}(z)}(n, (X, Y))\}_{n\in\mathbb{N}} \overset{c}{\approx} \{\textbf{Hyb}_{\pi_\cap^{\mathcal{F}_{\text{DL}}, \mathcal{F}_{\text{EqMask}}}, \mathcal{A}(z)}(n, (X, Y))\}_{n\in\mathbb{N}}.$$

Proof: Assume by contradiction the existence of an adversary $\mathcal{A}$ and a distinguisher D_{OPE} that distinguishes $\mathcal{A}$'s views for infinitely many n's, construct a distinguisher D_F that distinguishes F from a truly random function f that maps $\{0,1\}^m$ to $\mathbb{G}$, for infinitely many n's. Namely, upon given access to an oracle $\mathcal{O}$ that implements either F or f, and auxiliary input $(1^n, X, Y, z)$, D_F invokes $\mathcal{A}$ on $1^n, Y, z$. It then computes the first message of Protocol 5 as follows. It first invokes its oracle on the set $(0, \ldots, d)$; let $f_0, \ldots, f_d$ be the oracle's responses. D_F then creates polynomial $Q(\cdot) = (q_0, \ldots, q_d)$ and sends $\mathcal{A}$ the coefficients $\{g^{q_i} f_i\}_{i=0}^d$, for $g^{Q(\cdot)} = (g^{q_0}, \ldots, g^{q_d})$. Next, D_F emulates the trusted party for $\mathcal{F}_{\text{EqMask}}$, receiving the set the set $\{(y'_j, T_{y'_j})\}_{j\in[m_Y]}$ from $\mathcal{A}$ and replying with 1 for all $y'_j \in Z$ so that $T_{y'_j} = g^{Q(y'_j)}$, and with 0 otherwise. Finally, D_F invokes D_{OPE} on $\mathcal{A}$'s view and outputs whatever D_{OPE} does. Note that if oracle $\mathcal{O}$ implements function F, then $\mathcal{A}$'s view distributes as in the hybrid execution. On the other hand, if $\mathcal{O}$ implements f then $\mathcal{A}$'s view distributes as in the simulation (this is because the distribution on the coefficients is independent of the actual polynomial). Therefore, any gap in distinguishing these two views is immediately translated into a gap distinguishing F from f. $\square$

Finally, we prove the following claim.

SubClaim 4.4 *For any tuple of inputs (X,Y) and auxiliary input z,*

$$\{\mathbf{Hyb}_{\pi_\cap^{\mathcal{F}_{\mathrm{DL}},\mathcal{F}_{\mathrm{EqMask}}},\mathcal{A}(z)}(n,(X,Y))\}_{n\in\mathbb{N}} \overset{s}{\approx} \{\mathbf{HYBRID}_{\pi_\cap^{\mathcal{F}_{\mathrm{DL}},\mathcal{F}_{\mathrm{EqMask}}},\mathcal{A}(z)}(n,(X,Y))\}_{n\in\mathbb{N}}.$$

Proof: Formally, for every y'_j consider two cases. (**i**) $y'_j \notin X$ which implies that y'_j is not in the intersection and that $b_j = 0$ in the simulation of **Hyb**. Next, define a Bad event in which $\mathcal{A}$ receives $b_j = 1$ from the trusted party for $\mathcal{F}_{\mathrm{EqMask}}$ in the hybrid execution. Clearly, this event holds only if $T_{y'_j} = \mathsf{CFEval}(y'_j, K) = g^{R(y'_j)}$ for K the PRF key entered by the honest P_0, which implies that $\mathcal{A}$ must correctly guess $\mathsf{CFEval}(y'_j, K)$. We claim that the probability this event occurs is negligible due to the pseudorandomness of F and CFEval (in Section 5 we discuss the pseudorandomness of CFEval; see Theorem 5.1). Specifically, any successful guess with a non-negligible probability implies an attack on the PRF. Thus, the probability that Bad occurs is negligible. It therefore holds that the adversary's views are statistically close condition on the event that y'_j is not in the intersection. (**ii**) $y'_j \in X$ which implies that y'_j is in the intersection. Note that here there is no analogue bad event. This is because $b_j = 1$ only when $T_{y'_j} = C(y'_j) = \mathsf{CFEval}(y'_j, K)$, which implies that $b_j = 1$ in both executions due to correctness of $\mathcal{F}_{\mathrm{EqMask}}$.
$\square$

This concludes the proof. $\square$ $\blacksquare$

Efficiency. As in Protocol 2, the efficiency of Protocol 5 is dominated by the implementation of functionality $\mathcal{F}_{\mathrm{EqMask}}$. Our protocols from Section 3.3 can be easily modified to support this functionality without significantly affecting their overhead, since the parties can first compute the encryption of the closed form efficiency of the PRF and then compare it with the input of P_1. Therefore, the overall communication complexity is $O(m_X)$ group elements for sending the first message and $O(m_Y)$ (resp. $O(m_Y \log m_Y)$) group elements for the second phase of implementing $\mathcal{F}_{\mathrm{EqMask}}$ for each $y \in Y$, depending on the underlying PRF. In particular, the number of modular exponentiations implies multiplicative costs in the sets sizes since P_1 evaluates its masked polynomial for each element in Y. Next, we demonstrate how to reduce this cost using hash functions.

4.3 Improved Constructions Using Hash Functions

We now show how to reduce the computational overhead using hash functions by splitting the set elements into smaller bins. Our protocol is applicable for different hash functions such as: simple hashing, balanced allocations [ABKU99] and Cuckoo hashing [KMW08]. For simplicity, we first describe our protocol for the simple hashing case; see Section 4.3.3 for a discussion about extensions to the other two hashing. Informally, the parties first agree on a hash function that is picked from a family of hash functions and induces a set of bins with some upper bound on the number of elements in each bin. Next, P_0 maps its elements into these bins and generates a polynomial for each such bin, which is computed as in Protocol 5 but with a smaller degree. Finally, P_0 masks all the polynomials and sends them to P_1. Upon receiving the masked polynomials, P_1 maps its elements into the same set of bins and evaluates the masked polynomials for these mapped bins. In the last step, the parties unmask these evaluations.

Fixing some notations, we denote by h the hash function picked by the parties, by $\mathcal{B}$ the number of bins and by $\mathcal{M}$ the maximum number of elements allocated to any single bin (where $\mathcal{B}$ and $\mathcal{M}$ are parameters specified by the concrete hash function in use and further depend on m_X). Note that the potential number of allocated elements is bounded by $\mathcal{BM}$ which may be higher than the exact number m_X. This implies that the protocol must ensure that P_0 does not take advantage of that and introduce more set elements into the protocol execution. In addition, it must be ensured that a corrupted P_0 does not mask the zero polynomial, which would imply that P_1 accepts any value it substitutes in the masked polynomial. On the other hand, the protocol must ensure that a corrupted P_1 does not gain any information by entering incorrect values.

Next, we explain how the masking procedure is computed. Denote by $Q_j(\cdot)$ the polynomial associated with the jth bin. If the degree of $Q_j(\cdot)$ is smaller than $\mathcal{M}-1$ then P_0 fixes the values of the $\mathcal{M}_1-\deg(Q_j(\cdot))$ leading coefficients to be zeros. It then masks the ith coefficient of $Q_j(\cdot)$ by multiplying it with $\mathsf{PRF}_K((j-1)\cdot\mathcal{M}+i)$ for $i\in[0,\mathcal{M}-1]$. Furthermore, unmasking is computed by comparing the evaluation of the jth polynomial to the following computation

$$\prod_{i=0}^{j\mathcal{M}-1}\mathsf{PRF}_K(i)^{x^i}\Big/\prod_{i=0}^{(j-1)\mathcal{M}-1}\mathsf{PRF}_K(i)^{x^i}=\mathsf{PRF}_K((j-1)\mathcal{M})^{x^{(j-1)\mathcal{M}}}\cdot\ldots\cdot\mathsf{PRF}_K(j\mathcal{M}-1)^{x^{j\mathcal{M}-1}},$$

which is exactly the set of PRF values that mask polynomial $Q_j(\cdot)$.

More formally, our protocol uses two functionalities in order to ensure correctness. First, the parties call functionality $\mathcal{F}_{\mathrm{Bins}}$ for proving that the masked polynomials sent by P_0 are correctly defined. Namely,

Definition 4.5 $\mathcal{F}_{\mathrm{Bins}}:((K,\{C_j(\cdot)=(c_0^j,\ldots,c_{\mathcal{M}-1}^j)\}_{j\in[\mathcal{B}]}),m_X)\mapsto(-,b)$ *where* $b=1$ *only if none of the unmasked polynomials* $\{Q_j(\cdot)\}_j$ *with respect to PRF key* K *is the zero polynomial and their overall degrees is bounded by* m_X.

In addition, the parties call functionality $\mathcal{F}_{\mathrm{EqMaskHash}}$ in order to correctly unmask the polynomial evaluations $\{C_{h(y)}(y)\}_{y\in Y}$ for P_1.

Definition 4.6 $\mathcal{F}_{\mathrm{EqMaskHash}}:(K,\{y,h(y),C_{h(y)}(y)\}_{y\in Y})\mapsto(-,\{b_j\}_j)$, *where* $b_j=1$ *if* $C_{h(y)}(y)=\prod_{i=0}^{h(y)\mathcal{M}-1}\mathsf{PRF}_K(i)^{y^i}\Big/\prod_{i=0}^{(h(y)-1)\mathcal{M}-1}\mathsf{PRF}_K(i)^{y^i}$.

We continue with the detailed description of our set-intersection protocol in the hybrid model. In Sections 4.3.1 and 4.3.2 we discuss how to securely implement these functionalities.

Protocol 6 (Protocol $\pi_\cap$ with malicious security and hash functions.)

- **Input:** *Party* P_0 *is given a set* X *of size* m_X. *Party* P_1 *is given a set* Y *of size* m_Y. *Both parties are given a security parameter* 1^n.
- **The protocol:**
  1. **Fixing the parameters of the hash function.** *The parties fix the parameters* $\mathcal{B}$ *and* $\mathcal{M}$ *of the hash function and pick a hash function* $h:\{0,1\}^t\mapsto[\mathcal{B}]$. P_0 *invokes* $(K,param)\leftarrow\mathsf{KeyGen}(1^n,\mathcal{M}-1)$ *where* $param$ *includes a group description* $\mathbb{G}$ *of prime order* p *and a generator* g.
  2. **Masking the input polynomial.** *For every* $x\in X$, P_0 *maps* x *into bin* $h(x)$. *Let* $\mathcal{B}_j$ *denote the set of elements mapped into bin* j. *Next,* P_0 *constructs a polynomial* $Q_j(\cdot)=(q_0^j,\ldots,q_d^j)$ *with coefficients in* $\mathbb{Z}_p$ *and the set of roots* $\mathcal{B}_j$. *If* $|\mathcal{B}_j|<\mathcal{M}$, P_0 *fixes the leading* $\mathcal{M}-|\mathcal{B}_j|-1$ *coefficients to zero.*

     For each bin $j\in[\mathcal{B}]$, P_0 *defines a new* $(\mathcal{M}-1)$*-degree polynomial* $\widetilde{R}_j(\cdot)=(\tilde{r}_0^j,\ldots,\tilde{r}_{\mathcal{M}-1}^j)$ *over* $\mathbb{G}$, *where* $\tilde{r}_i^j$ *is defined by* $\mathsf{PRF}_K((j-1)\mathcal{M}+i)$ *for all* $i\in[0,\mathcal{M}-1]$. P_0 *sends* P_1 $param$ *and the masked polynomials* $\{C_j(\cdot)\}_j=\{g^{q_0^j}\tilde{r}_0^j,\ldots,\ldots,g^{q_{\mathcal{M}-1}^j}\tilde{r}_{\mathcal{M}-1}^j\}_j$, *where multiplication is implemented in* $\mathbb{G}$.

     P_0 *further proves the knowledge of the discrete logarithm of* $c_i^j=g^{q_i^j}\tilde{r}_i^j$ *for all* i *and* j *with respect to a generator* g, *by invoking an ideal execution of* $\mathcal{F}_{\mathrm{DL}}$ *on input* $\{((g,c_i^j),\log_g c_i^j)\}_{i\in[0,\mathcal{M}-1],j\in[\mathcal{B}]}$.[9] *The*

[9] See Footnote 7.

input of P_1 for $\mathcal{F}_{\rm DL}$ is $\{(g, c_i^j)\}_{i\in[0,\mathcal{M}-1],j\in[\mathcal{B}]}$.

Finally, P_0 proves correctness using $\mathcal{F}_{\rm Bins}$ where P_0 enters K and P_1 enters the masked polynomials.

3. **Unmasking the result.** *Upon receiving the masked polynomials $\{C_j(\cdot) = (c_0^j, \ldots, c_{\mathcal{M}-1}^j)\}_{j\in[\mathcal{B}]}$ and upon receiving accepting messages from $\mathcal{F}_{\rm DL}, \mathcal{F}_{\rm Bins}$, party P_1 computes the following for every $y \in Y$ (picked in a random order). It first maps y into bin $h(y)$ and then computes the polynomial evaluation $C_{h(y)}(y) = \prod_{i=(h(y)-1)\mathcal{M}}^{h(y)\mathcal{M}-1}(c_i^{h(y)})^{y^i}$. I.e., $C_{h(y)}(\cdot)$ is evaluated* in the exponent.

   Next, the parties invoke an ideal execution of $\mathcal{F}_{\rm EqMaskHash}$, where the input of P_0 is K and the input of P_1 is the set $\{(y, h(y), C_{h(y)}(y))\}_{y\in Y}$.

   P_1 outputs y only if the output from $\mathcal{F}_{\rm EqMaskHash}$ on $(y, h(y), C_{h(y)}(y))$ is 1.

Theorem 4.7 *Protocol 6 securely realizes functionality $\mathcal{F}_\cap$ in the presence of malicious adversaries in the $\{\mathcal{F}_{\rm DL}, \mathcal{F}_{\rm Bins}, \mathcal{F}_{\rm EqMaskHash}\}$-hybrid model.*

Security follows easily from the secure implementations of $\mathcal{F}_{\rm Bins}$ and $\mathcal{F}_{\rm EqMaskHash}$ and the proof of Protocol 5. We discuss these protocols next. We stress that P_1 needs to ensure in Protocol 6 that P_0 indeed uses the same PRF key for both sub-protocols (for instance by ensuring that P_0 enters the same commitment of K).

4.3.1 Realizing $\mathcal{F}_{\rm Bins}$

In this section we design a protocol $\pi_{\rm Bins}$ for securely implementing functionality $\mathcal{F}_{\rm Bins}$. To prove that none of the polynomials is the all zeros polynomial we evaluate each masked polynomial on a random element and then verify that the result is different than zero. In particular, for each j the parties first agree on a random element z_j and then compute the polynomial evaluation $C_j(z_j)$. Next, the parties verify whether $C_j(z_j) = \widetilde{R}_j(z_j)$ where $\widetilde{R}_j(\cdot)$ is the masking polynomial of $C_j(\cdot)$. Note that if $Q_j(\cdot)$ is not the all zeros polynomial then $C_j(z_j) \neq \widetilde{R}_j(z_j)$ with overwhelming probability over the choice of z_j. This is because there exists a coefficient $q_{i,j} \neq 0$ which implies that for $C_j(z_j) = Q_j(z_j) \cdot \widetilde{R}_j(z_j)$. Now since $Q_j(z_j) \neq 0$ it holds that $C_j(z_j) \neq \widetilde{R}_j(z_j)$. On the other hand, in case $Q_j(\cdot)$ is the zero polynomial then it holds that $C_j(z_j) = \widetilde{R}_j(z_j)$ for all z_j. This is because $Q_j(z_j) = 0$ as all its coefficients equal zero.

The more challenging part is to prove that the overall degrees of all polynomials $\{Q_j(\cdot)\}_j$ is bounded by m_X. Our proof ensures that as follows. First, P_0 picks a PRF key K and forwards P_1 a commitment of K together with encryptions of $f = \big(f_0 = \mathsf{PRF}_K(0), \ldots, f_{\mathcal{BM}-1} = \mathsf{PRF}_K(\mathcal{BM}-1)\big)$ (that are encrypted using the El Gamal encryption scheme; see Section 2.4). Next, P_0 proves that it computed the sequence f correctly. This can be achieved by exploiting the closed form efficiency property of the PRF. Namely, the parties mutually compute the encryption of $\prod_{i=0}^{\mathcal{BM}-1}\mathsf{PRF}_K(i)^{z^i}$ for some random z, and then compare it with the encryption of $\prod_{i=0}^{\mathcal{BM}-1} f_i^{z^i}$. In particular, the latter computation is carried out on the ciphertexts that encrypt the corresponding values from f by utilizing the homomorphic property of El Gamal. Then, equality is verified such that P_0 proves that the two ciphertexts encrypt the same value. Finally, the parties divide the vector of ciphertexts f with the polynomials coefficients $\{C_j(\cdot)\}_{j\in[\mathcal{B}]}$ component-wise (note that both vectors have the same length). P_0 then proves that the overall degrees of the polynomials is as required using a sequence of zero-knowledge proofs. The last part of our proof borrows ideas from [HN12]. We continue with the formal description of our protocol.

Protocol 7 (Protocol $\pi_{\rm Bins}$ with malicious security.)

- **Input:** *Party P_0 is given a PRF key K for function* PRF. *Both parties are given a security parameter 1^n, masked polynomials $\{C_j(\cdot) = (c_0^j, \ldots, c_{\mathcal{M}-1}^j)\}_{j\in[\mathcal{B}]}$, $(\mathbb{G}, p, g)$ for a group description $\mathbb{G}$ of prime order p and a generator g, and an integer m_X.*
- **The protocol:**

1. **Setup.** *P_0 generates $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^n)$ for the El Gamal encryption scheme for group $\mathbb{G}$. It then computes the set $f = \big(f_0 = \mathsf{PRF}_K(0), \ldots, f_{\mathcal{BM}-1} = \mathsf{PRF}_K(\mathcal{BM}-1)\big)$ and sends to P_1 their encryptions under PK, denoted by $(e_0, \ldots, e_{\mathcal{BM}-1})$, as well as PK.*
2. **Proving the correctness of f.** *The parties pick $z \leftarrow \mathbb{Z}_p$ at random and then compute $e_f = \prod_{i=0}^{\mathcal{MB}-1} e_i^{z^i}$. Next, the parties compute the encryption of the product $\prod_{i=0}^{\mathcal{BM}-1} \mathsf{PRF}_K(i)^{z^i}$, denoted by e_{PRF}, which corresponds to the closed form efficiency function of PRF. Finally, P_0 proves that the two ciphertexts encrypt the same plaintext by proving that e_f/e_{PRF} is a Diffie-Hellman tuple using $\pi_{\rm DL}$ (see Section 2.5).*
3. **Proving a bound m_X on the overall degrees.** *If $\pi_{\rm DL}$ is verified correctly, the parties compute the differences with respect to the masked polynomials $\{C_j(\cdot)\}_j$ and plaintexts f, component-wise. Namely, for all $j \in [\mathcal{B}]$ and $i \in [0, \mathcal{M}-1]$ the parties compute the encryption of $c_i^j/f_{(j-1)\mathcal{M}+i}$. We denote the result vector of ciphertexts by $c_{\rm DIFF}$.*

   P_0 then sets $Z_{i,j} = 1$ for $0 \le i \le \deg(Q_j(\cdot))$, and otherwise $Z_{i,j} = 0$. P_0 computes $z_{i,j} = \mathsf{Enc}_{\rm PK}(Z_{i,j})$ and sends $\{z_{i,j}\}_{i,j}$ to P_1. P_0 proves that $Z_{0,j}, Z_{1,j}, \ldots, Z_{M-1,j}$ is monotonically non-increasing. For that, P_0 and P_1 compute encryptions of $Z_{i,j} - Z_{i+1,j}$ and $Z_{i,j} - Z_{i+1,j} - 1$, and P_0 proves that $Z_{i,j} - Z_{i+1,j} \in \{0, 1\}$ by showing that one of these encryptions denotes a Diffie-Hellman tuple using $\pi_{\rm DDH}$.

   P_0 completes the proof that the values $Z_{i,j}$ were constructed correctly by proving for all i, j that one of the encryptions $\{e_{(j-1)\mathcal{M}+i}, z'_{i,j}\}$ is an encryption of zero, where $z'_{i,j}$ is an encryption of $1 - Z_{i,j}$.[10]

   Finally, to prove that the sum of degrees of the polynomials $\{Q_j(\cdot)\}$ equals m_X, both parties compute an encryption τ of $T = \sum_{i,j} Z_{i,j} - m_X$. Then P proves that $(\mathrm{PK}, \mathsf{Enc}_{\rm PK}(T))$ is a Diffie-Hellman tuple using $\pi_{\rm DDH}$.
4. **Checking zero polynomials.** *If all the proofs are verified correctly, then for any $j \in [\mathcal{B}]$ the parties compute $C_j(z_j)$ where $z_j \leftarrow \mathbb{Z}_p$. The parties call $\mathcal{F}_{\rm EqMaskHash}$ with inputs $(K, \{z_j, j, C_j(z_j)\}_{j\in[\mathcal{B}]})$. Let $\{b_j\}_{j\in[\mathcal{B}]}$ be P_1's output from this ideal call.*[11]
5. *P_1 outputs $b = 1$ only if $b_j = 0$ for all j.*

Theorem 4.8 *Assume that El Gamal is IND-CPA, then Protocol 7 securely realizes functionality $\mathcal{F}_{\rm Bins}$ in the presence of malicious adversaries in the $\{\mathcal{F}_{\rm DL}, \mathcal{F}_{\rm DDH}, \mathcal{F}_{\rm EqMaskHash}\}$-hybrid model.*

Proof Sketch: Security is easily stated due to the security of the zero-knowledge proofs and functionality $\mathcal{F}_{\rm EqMaskHash}$. Specifically, the protocol enables P_1 to verify the following two properties on the masked polynomials $\{C_j(\cdot) = (c_0^j, \ldots, c_{\mathcal{M}-1}^j)\}_{j\in[\mathcal{B}]}$. First, that the overall number of polynomial degrees is m_X, which implies that P_0 used at most m_X set elements when creating these polynomials. This is verified by ensuring that f was correctly constructed and by the soundness of the zero knowledge proofs $\pi_{\rm DL}$ and $\pi_{\rm DDH}$. Specifically, if one of the elements in f is incorrect then P_0 will be caught with all but negligible probability. Denote by c_i the ciphertext that encrypts f_i and say that $f_\tau \neq \mathsf{PRF}_K(\tau)$ for some $\tau \in [0, ..., \mathcal{BM}-1]$. Then the probability that

$$\prod_{i=0}^{\mathcal{BM}-1} \mathsf{PRF}_K(i)^{z^i} = \prod_{i=0}^{\mathcal{BM}-1} f_i^{z^i}$$

implies the probability that

$$f_\tau = \left(\prod_{i=0}^{\mathcal{BM}-1} \mathsf{PRF}_K(i)^{z^i} - \prod_{i=0, i\neq\tau}^{\mathcal{BM}-1} f_i^{z^i} \right)^{\left(z^\tau\right)^{-1}}$$

which holds with negligible probability due to the random choice of z. Next, in Step 3, P_0 proves that it constructed the sequence of ciphertexts $z_{i,j}$ correctly by demonstrating that non-zero values are always mapped to ciphertexts that encrypt 1 and that this sequence is a monotonically non-increasing.

[10]We wish to avoid the case where $e_{(j-1)\mathcal{M}+i}$ is an encryption of a non-zero value while $z'_{i,j}$ encrypts zero.

[11]Note that z_j may be an element that is not mapped to the jth bin.

Second, P_0 proves that none of the masked polynomials is the zero polynomial. This is shown by substituting a random element z_j in $C_j(\cdot)$. Note that z_j is a root of $C_j(\cdot)$ only with negligible probability. Therefore, the probability that $C_j(z_j) = 0$ condition on the even that z_j is not a root of $C_j(\cdot)$ is also negligible due to the random choice of z_j. $\blacksquare$

The efficiency of our protocol is dominated by Steps 2 and 4, where in the former step the parties compute the closed form efficiency relative to the set f in time $O(\mathcal{BM}) = O(m_X)$ and in the latter step the parties substitute a random element in every polynomial C_j. Overall, the overhead of this step relative to PRF PRF_1 implies $O(\mathcal{B}) = O(m_X)$ group elements and modular exponentiations. For PRF PRF_2 this step implies $O(\mathcal{B}\log m_X) = O((m_X \backslash \log\log m_X) \cdot \log m_X)$ cost; see a discussion below.

4.3.2 Realizing $\mathcal{F}_{\text{EqMaskHash}}$

The next protocol is designed in order to compare the result of P_1's polynomial evaluations on the set Y with the masking polynomials. Basically, for every $y \in Y$, P_1 computes first $C_{h(y)}(y)$ and the parties then run a protocol for comparing $\{C_{h(y)}(y)\}_{y \in Y}$ with $\{\widetilde{R}_{h(y)}(y)\}_{y \in Y}$. Finally, for every j, P_1 outputs 1 only if equality holds. The actual implementation of this functionality depends on the underlying PRF. We consider two different implementations here. First, considering our protocol from Section 4.3 designed for PRF_1, an analogue protocol can be designed with the modification that the parties now compare $C_{h(y)}(y)$ against

$$g^{\frac{k_0\left((k_1 y)^{(h(y)+1)\mathcal{M}-1} - (k_1 y)^{h(y)\mathcal{M}-1}\right)}{k_1 y - 1}}$$

where $h(y)$ is only known to P_1. Note that our protocol from Section 3.3 does not need to rely on the fact that both parties know the polynomial degree d for computing this formula. It is sufficient to prove that the computation of some ciphertext c to the power of $h(y)$ is consistent with a ciphertext encrypting $g^{h(y)}$, where such a ciphertext can be provided by P_1. See the protocol from Section 3.3 and the ZK proof π_{EQ} for more details. The overall overhead of the modified protocol is also constant.

Next, considering the unmasking protocol for function PRF_2, the parties compute the following formula that corresponds to the masking of the polynomial that is associated with bin $h(y)$,

$$g^{k_0\left(1+k_1,y\right)\left(1+k_2 y^2\right)\ldots\left(1+k_m y^{2^{\log(h(y)\mathcal{M}-1)}}\right)} \Big/ g^{k_0\left(1+k_1,y\right)\left(1+k_2 y^2\right)\ldots\left(1+k_m y^{2^{2^{\log((h(y)-1)\mathcal{M}-1)}}}\right)}.$$

Note that computing this formula requires $O(\log m_X)$ exponentiations on the worst case if the bin number implies a high value so that $h(y)\mathcal{M}$, which determines the polynomial degree, is $O(m_X)$.

Security is informally stated as follows. If P_0 is corrupted then security follows similarly to the security argument of the protocols implementing $\mathcal{F}_{\text{MaskPoly}}$ (Section 3.3) since P_0 enters the same input for both functionalities and runs the same computations with respect to its PRF key. The interesting and less trivial corruption case is of P_1. We consider two bad events here: (1) A corrupted P_1 enters y, h' for which $h' \neq h(y)$. This implies that the parties will not compute the correct unmasking. (2) A corrupted P_1 enters consistent $y, h(y)$, but an incorrect value $C_{h(y)}(y)$. Note that upon extracting P_1's input to the protocol execution, the simulator can always tell whether this input corresponds to the first or the second case, or neither. Specifically, in the first case the parties compute the unmasking on y for which element y in not allocated to the specified bin h'. This implies that P_1 would always obtain 0 from the protocol execution unless it correctly guesses $\widetilde{R}_{h'}(y)$, which only occurs with a negligible probability due to the pseudorandomness of the PRF. Therefore we can successfully simulate this case by always returning zero. Moreover, the security argument of the latter case boils down to the argument presented in the proof for a single polynomial shown in the proof of Theorem 4.4, since in this case P_1 enters $h(y)$ that is consistent with y, therefore the parties compute the correct masking for y.

4.3.3 Using More than One Hash Function

In some cases, such as for balanced allocation hash function [ABKU99], better performance are obtained by using a pair of hash functions h_1, h_2, which allocate elements into two distinct bins. That is, the input to the functionality is defined by $(K, \{y, h_1(y), h_2(y), C_{h(y)}(y)\}_{y \in Y}) \mapsto (-, \{b_j\}_j)$. This poses a problem in our setting since a corrupted P_1 may deviate from the protocol by substituting a different element with respect to each hash function, and learn some information about P_0's input. Specifically, if P_1 learns that some element $y \in X$ was not allocated to $h_1(y)$ it can conclude that P_0 has $\mathcal{M}$ additional elements that are already mapped into bin $h_1(y)$. Note that this leaked information cannot be simulated since it depends on the real input X. In this case we need to verify that P_1 indeed maps the same element into both bins correctly. A simple observation shows that if this is not the case then the simulation fails only for elements that are in the intersection. Meaning, there exists a bin for which the membership result is positive (since otherwise the adversary anyway learns 0, and it cannot distinct the cases of non-membership and incorrect behaviour). We thus define the polynomials slightly different, forcing correct behaviour.

Specifically, the polynomial $Q_j(\cdot)$ that is associated with the set of elements $\mathcal{B}_j$ (namely, the elements that are mapped to the jth bin) is defined as follows. For each $x \in \mathcal{B}_j$, set $Q_j(x) = g^{h_1(x) \cdot h_2(x)}$ where $h_1(x)$ and $h_2(x)$ are viewed as elements in $\mathbb{Z}_p$. Next, in the unmasking phase, for any tuple (y, h_1, h_2, C_y) entered by P_1, the parties compare C_y with both $\left(\prod_{i=0}^{h_1\mathcal{M}-1} \mathsf{PRF}_K(i)^{y^i} \Big/ \prod_{i=0}^{(h_1-1)\mathcal{M}-1} \mathsf{PRF}_K(i)^{y^i} \right) \cdot g^{h_1 \cdot h_2}$ and $\left(\prod_{i=0}^{h_2\mathcal{M}-1} \mathsf{PRF}_K(i)^{y^i} \Big/ \prod_{i=0}^{(h_2-1)\mathcal{M}-1} \mathsf{PRF}_K(i)^{y^i} \right) \cdot g^{h_1 \cdot h_2}$ such that the functionality returns 1 to P_1 if equality holds with respect to one of the comparisons. Therefore, P_1 will learn that an element $y \in X$ only if it entered h_1 and h_2 such that $h_1 + h_2 = h_1(y) + h_2(y)$. Note that this implies that if one of the h_1, h_2 values is inconsistent with $h_1(y), h_2(y)$ yet equality holds, then the other value is also inconsistent with high probability. In this case, P_1's output will always be 0 since the incorrect polynomials will be unmasked.

We further need to claim that for any $y \notin X$ the protocol returns 0 with overwhelming probability. Specifically, we need to claim that the probability that either $Q_{h_1(y)}(y) = g^{h_1(y)+h_2(y)}$ or $Q_{h_2(y)}(y) = g^{h_1(y)+h_2(y)}$, is negligible. In order to simplify our argument, we slightly modify our construction and fix $Q_j(x) = \mathsf{PRF}_K(g^{h_1(x)+h_2(x)})$ for any $x \in \mathcal{B}_j$ using a PRF K key that is mutually picked by both parties. In this case, we can easily claim that the probability that the protocol returns 1 for $y \notin X$ is negligible since that implies that either $Q_{h_1(y)}(y)$ or $Q_{h_2(y)}(y)$ equal the pseudorandom value $\mathsf{PRF}_K(g^{h_1(y)+h_2(y)})$ for $y \notin X$. We stress that the PRF key for this purpose can be publicly known since pseudorandomness is still maintained as long as the algorithm for generating the bin polynomials does not use this key. We further note that both algebraic PRFs that we consider in this paper can be easily evaluated over an encrypted plaintext given the PRF key since they only require linear operations.

Finally, a similar solution can be easily adapted for Cuckoo hashing with a stash [KMW08] (by treating the stash as a third polynomial). Nevertheless, Cuckoo hashing using a stash suffers from the following drawback. It has been proven in [KMW08] that for any constant s, using a stash of size s implies an overflow with probability $O(n^s)$ (taken over the choice of the hash functions). Specifically, if the algorithm aborts whenever the original choice of hash functions results in more than s items being moved to the stash, then this means that the algorithm aborts with probability of at most $O(n^s)$. Consequently, P_1 can identify with that probability whether a specific potential input of P_0 does not agree with the hash functions h_1 and h_2. This probability is low but not negligible. On the other hand, Broder and Mitzenmacher [BM01] have shown for balanced allocations hash function that asymptotically, when mapping n items into n bins, the number of bins with i or more items falls approximately like $2^{2.6i}$. This means that if $\mathcal{M} = \omega(\log \log n)$ then except with negligible probability no bin will be of size greater than $\mathcal{M}$. Nevertheless, (ignoring the abort probability), Cuckoo hashing performs better than balanced allocation hash functions, and by tuning the parameters accordingly this abort probability can be ignored for most practical applications.

Functionality $\mathcal{F}_{\text{CPRF}}$

Functionality $\mathcal{F}_{\text{CPRF}}$ communicates with with parties P_0 and P_1, and adversary $\mathcal{S}$.

1. Upon receiving a message (key, K) from P_0, send message key to $\mathcal{S}$ and record K.
2. Upon receiving (input, x) from P_1, send message input to adversary $\mathcal{S}$. Upon receiving an approve message, send $\mathsf{PRF}_K(x)$ to P_1. Otherwise, send $\perp$ to P_1 and abort.

Figure 1: The committed oblivious PRF evaluation functionality.

4.3.4 Efficiency

The efficiency of our protocol depends on the parameters $\mathcal{B} = O(m_X / \log\log m_X)$ and $\mathcal{M} = O(\log\log m_X)$ that are specified by the underlying hash function, as well as the PRF implementation that induces the overhead of the implementations of $\mathcal{F}_{\text{Bins}}$ and $\mathcal{F}_{\text{EqMaskHash}}$. Concretely, when implementing the algebraic PRF with PRF_1 the number of exponentiations computed by the parties is $O(\mathcal{BM} + m_Y\mathcal{M}) = O(m_X + m_Y \log\log m_X)$, whereas the number of transmitted group elements is $O(\mathcal{BM} + m_Y) = O(m_X + m_Y)$. Moreover, implementing the algebraic PRF using PRF_2 implies the overhead of $O(m_X + m_Y \log m_X)$ exponentiations and the communication is as above.

5 Committed Oblivious PRF Evaluation

The oblivious PRF evaluation functionality $\mathcal{F}_{\text{PRF}}$ is an important functionality that is defined by $(K, x) \mapsto (-, \mathsf{PRF}_K(x))$. One known example for a protocol that implements $\mathcal{F}_{\text{PRF}}$ is the instantiation based on the Naor-Reingold pseudorandom function [NR97] (specified in Section 3.1.2), that is implemented by the protocol presented in [FIPR05] (and proven secure in the malicious setting in [HL10a]). This protocol involves executing an oblivious transfer for every bit of the input x. Nevertheless, it has major drawback since it does not enforce the usage of *the same* key for multiple evaluations, which is often required. In this section, we observe first that the algebraic closed form efficiency of PRFs PRF_1 and PRF_2, specified in Section 3.1, are PRFs as well. Moreover, the protocols for securely evaluating these functions induce efficient implementations for the *committed* oblivious PRF evaluation functionality with respect to these new PRFs in the presence of adaptive inputs. This is because the PRF evaluations protocols are implemented with respect to the same set of key commitments. We formally define this functionality in Figure 1.

More formally, let PRF be an algebraic PRF from a domain $\{0,1\}^m$ into a group $\mathbb{G}$. Then define the new function $\mathsf{PRF}' : \mathbb{Z}_p \mapsto \mathbb{G}$ by $\mathsf{PRF}'_K(x) = \prod_{i=0}^{l} [\mathsf{PRF}_K(i)]^{x^i}$. Note that the domain size of PRF' must be bounded by $l+1$, since upon observing $l+1$ evaluations of PRF' it is possible to interpolate the coefficients of the polynomial $\{\mathsf{PRF}_K(i)\}_i$ (in the exponent). On the other hand, it is easy to verify that if l is polynomial in the security parameter then PRF' is a PRF. Since the domain size is upper bounded by $l+1$, this means that the domain size must be polynomial.

Theorem 5.1 *Assume that* $\mathsf{PRF} : \{0,1\}^m \mapsto \mathbb{G}$ *is a PRF, then* PRF' *is a PRF for l that is polynomial in the security parameter.*

Proof: We prove security by a reduction to the pseudorandomness of PRF. Namely, assume by contradiction a distinguisher $D_{F'}$ that distinguishes PRF' from a truly random function. We construct a distinguisher D_F that distinguishes PRF from a truly random function. Specifically, consider a reduction in which a distinguisher D_F communicates with an oracle $\mathcal{O}$ on the set of queries $[0, l]$. Upon receiving the oracle's responses $f_0, \dots, f_l$, D_F computes $\prod_{i=0}^{l} f_i^{x^i}$ for any query $x \in \mathbb{Z}_p$ made by $D_{F'}$, as long as the number of distinct queries is bounded by $l+1$. Finally, D_F outputs whatever $D_{F'}$ does. Clearly, if $D_{F'}$ breaks the security of PRF' then D_F breaks the security of PRF. ■

We implement PRF' using the two PRFs from Section 3.1.1 and obtain two new PRF constructions under the strong-DDH and DDH assumptions. Let $K = (k_0, k_1)$ be the key for the PRF PRF_1 with the strong-DDH based security, and recall that the closed form efficiency for this function is defined by

$$\mathsf{PRF}'_K(x) = \mathsf{CFEval}_h(x, K) = g^{\frac{k_0(k_1^{d+1}x^{d+1}-1)}{k_1x-1}}.$$

This implies that PRF' only requires a constant number of modular exponentiations. See Section 3.3 for secure implementations of obliviously evaluating PRF'. Next, let $K = (k_0, \dots, k_m)$ be the key for the Naor-Reingold PRF, and recall that the closed form efficiency of this function is defined by

$$\mathsf{PRF}'_K(x) = \mathsf{CFEval}_{h,z}(x, K) = g^{k_0(1+k_1,x)(1+k_2x^2)\dots(1+k_mx^{2^m})}$$

which requires $O(\log l) = O(m)$ operations, namely, a logarithmic number of operations in the domain size where x is an m-bits string. This is the same order of overhead induced by the [FIPR05] implementation that requires an OT for each input bit. Nevertheless, our construction has the advantage that it also achieves easily the property of a committed key since multiple evaluations can be computed with respect to the same committed PRF key. Plugging-in our protocol inside the protocols for keyword search, OT with adaptive queries [FIPR05] and set-intersection [HL10a] implies security against malicious adversaries fairly immediately. It is further useful for search functionalities as demonstrated below.

5.1 The Set-Intersection Protocol

We continue with describing our set-intersection protocol. Informally, P_0 generates a PRF key for PRF and evaluates this function on its set X. It then sends the evaluation results to P_1 and the parties engage in a committed oblivious PRF protocol that evaluates PRF on the set Y. P_1 then concludes the intersection. In order to handle a technicality in the security proof, we require that P_0 must generate its PRF key *independently* of its input X, since otherwise it may maliciously pick a secret key that implies collisions on elements from X and Y, causing the simulation to fail. We ensure key independence by asking the parties to mutually generate the PRF key after P_0 has committed to its input. Then upon choosing the PRF key, the parties invoke two variations of functionality $\mathcal{F}_{\mathrm{CPRF}}$, denoted by $\mathcal{F}^0_{\mathrm{CPRF}}$ and $\mathcal{F}^1_{\mathrm{CPRF}}$ defined below.

Definition 5.2 *Let X and Y be subsets of a predetermined arbitrary domain $\{0,1\}^t$ and m_X and m_Y the respective upper bounds on the sizes of X and Y. Then we define functionality $\mathcal{F}^0_{\mathrm{CPRF}}$ as follows:*

$$((K, (x_1, \dots, x_{m_X}), R), (c_{\mathrm{KEY}}, (c_1, \dots, c_{m_X}), \mathrm{PK})) \mapsto (-, (\mathsf{PRF}_K(x_1), \dots, \mathsf{PRF}_K(x_{m_X})))$$

if c_i encrypts x_i for all i and c_{KEY} is a commitment of K where verification is carried out using the randomness R.

In the final step of the set-intersection protocol the parties call functionality $\mathcal{F}^1_{\mathrm{CPRF}}$ in order to evaluate the PRF on the set Y.

Definition 5.3 *Let X and Y be subsets of a predetermined arbitrary domain $\{0,1\}^t$ and m_X and m_Y the respective upper bounds on the sizes of X and Y. Then we define functionality $\mathcal{F}^1_{\mathrm{CPRF}}$ as follows:*

$$((K, R), (c_{\mathrm{KEY}}, (y_1, \dots, y_{m_Y}))) \mapsto (-, (\mathsf{PRF}_K(y_1), \dots, \mathsf{PRF}_K(y_{m_Y})))$$

if c_{KEY} is a commitment of K where verification is carried out using the randomness R.

In both executions the output is given to P_1 that computes the intersection of the results.

5.1.1 Realizing $\mathcal{F}^0_{\rm CPRF}$ and $\mathcal{F}^1_{\rm CPRF}$

Implantation-wise, there is not much of a difference between the protocols for the two functionalities, which mainly differ due to the identity of the party that enters the input values to the PRF (where the same committed key is used for both protocol executions). We note that the realizations of $\mathcal{F}^0_{\rm CPRF}$ and $\mathcal{F}^1_{\rm CPRF}$ can be carried out securely based on the implementations of the closed form efficiency functions shown in Section 3.3, since our committed PRFs are based on these functions. More concretely, the difference with respect to functionality $\mathcal{F}^0_{\rm CPRF}$ is that now when P_0 is corrupted the simulator needs to extract the randomness used for committing the PRF key and the x_i's elements which can be achieved using the proof of knowledge $\pi_{\rm 2DL}$ since the parties use the El Gamal PKE. Next, the parties continue with the PRF evaluations where the ZK proofs are carried out with respect to the same key commitment. Finally, the implementation of $\mathcal{F}^1_{\rm CPRF}$ follows similarly but without the additional proof we added above for $\mathcal{F}^0_{\rm CPRF}$ in order to extract the randomness of the committed input of P_0. We next describe a detailed protocol that securely realizes $\mathcal{F}^1_{\rm CPRF}$ for PRF PRF_1 from Section 3.1.1, which is very similar to Protocol 3 from Section 3.3 and is based on the same building blocks. A protocol realizing $\mathcal{F}^0_{\rm CPRF}$ follows analogously, except that P_0 performs the computations based on $(x_1, \ldots, x_{m_X})$ and proves consistency with the commitments that P_1 holds.

Protocol 8 (Protocol $\pi^1_{\rm CPRF}$ with malicious security.)

- **Input:** *Party P_0 is given a PRF key $K = (k_0, k_1)$ and randomness R. Party P_1 is given $(c_{\rm KEY}, (y_1, \ldots, y_{m_Y}), {\rm PK})$. Both parties are given a security parameter 1^n and $(\mathbb{G}, p, g)$ for a group description $\mathbb{G}$ of prime order p and a generator g.*
- **The protocol:**
  1. **Proving the knowledge of R.** *P_0 and P_1 engage in a pair of parallel executions of $\pi_{\rm 2DL}$ for which P_0 proves the knowledge of $c_{\rm KEY}$. Note that $c_{\rm KEY}$ is a pair of ciphertexts (c_{k_1}, c_{k_2}) that respectively encrypts (k_1, k_2) and that given an El Gamal ciphertext $\langle c_1, c_2\rangle = \langle g^r, h^r g^m\rangle$, the proven statement is c_2 for which P_0 proves that it knows r and m.*
  2. **Generating additively homomorphic keys.** *P_0 runs $({\rm PK}', {\rm SK}') \leftarrow \mathsf{Gen}'(1^n)$ and sends ${\rm PK}'$ to P_1, as well as an encryption of k_1, denoted by $c'_{k_1} = \mathsf{Enc}'_{pk'}(k_1)$. P_0 proves consistency with c_{k_1} using $\pi_{\rm Eq-Exp}$. Next, the following steps are performed for any element from the set $(y_1, \ldots, y_{m_Y})$ in parallel.*
  3. **Computing encryption of (masked) $\mathbf{k_1 y_i - 1}$.** *P_1 responds with $c' = \mathsf{Enc}'_{\rm PK'}\big((k_1 y_i - 1)\cdot r\big)$ and $c_r = \mathsf{Enc}_{\rm PK}(r)$ for some random mask r.*[12]
  4. **Computing encryption of $\mathbf{(k_1 y_i - 1)^{-1}}$.** *Upon receiving ciphertext c' from P_1, P_0 decrypts it and re-encrypts it, as well as the inverse of this plaintext modulo p, under ${\rm PK}$, creating ciphertexts $c, c_{\rm INV}$ and proving consistency between c' and c using $\pi_{\rm Eq-Exp}$, and between c and $(c_{\rm INV}, \mathsf{Enc}_{\rm PK}(g^1))$ using $\pi_{\rm MULT}$, where the latter ciphertext is generated by P_0 and proven correct using $\pi_{\rm DDH}$.*

     As a response, P_1 unmasks r from the plaintext within ciphertext c by raising it to the power of $r \bmod p$, creating ciphertext $c_{r^{-1}}$ and proving consistency with c_r using $\pi_{\rm EQ}$.
  5. **Computing encryptions of $\mathbf{k_1^{d+1}}$ and $\mathbf{y_i^{d+1}}$.** *P_0 computes an encryption of k_1^{d+1}, denoted by $c_{k_1^{d+1}}$, and proves consistency between g^{d+1} and $c_{k_1^{d+1}}$ using $\pi_{\rm EQ}$. Similarly, P_0 computes the encryption of y_i^{d+1}, denoted by $c_{y_i^{d+1}}$, and proves correctness (where an encryption of y_i can be recovered from the encryption of $k_1 y_i - 1$, denoted above by $c_{r^{-1}}$).*
  6. **Computing encryptions of $\mathbf{k_0(k_1 y_i - 1)^{-1}}$ and $\mathbf{k_0 k_1^{d+1}(k_1 y_i - 1)^{-1}}$.** *Given ciphertexts $c_{\rm INV}$, $c_{k_1^{d+1}}$ and c_{k_0}, P_0 computes the encryptions of $k_0(k_1 y_i - 1)^{-1}$ and $k_0 k_1^{d+1}(k_1 y_i - 1)^{-1}$ and proves consistency relative to $c_{\rm INV}$, $c_{k_1^{d+1}}$ and c_{k_0} using $\pi_{\rm EQ}$ (where the proof of the later computation involves running $\pi_{\rm EQ}$ twice). Let c_0 and c'_0 denote the respective outcomes.*

[12] See Footnote 4.

7. **Computing encryption of $\mathbf{k_0 k_1^{d+1} y_i^{d+1}(k_1 y_i - 1)^{-1}}$.** *Given ciphertexts $c_{y_i^{d+1}}$ and c'_0, P_1 computes the encryption of $k_0 k_1^{d+1} y_i^{d+1}(k_1 y_i - 1)^{-1}$ and proves consistency using $\pi_{\rm EQ}$. Let c_1 denote the respective outcome.*
8. **Outcome.** *Finally, the parties decrypt c_1/c_0 for P_1 by running $\pi_{\sf Dec}$, that outputs the result.*

Theorem 5.4 *Assume $\Pi = (\pi_{\rm KeyGen}, {\sf Enc}, \pi_{\rm DEC})$, $\Pi' = ({\sf Gen}', {\sf Enc}', {\sf Dec}')$ are as in Section 3.3, then Protocol 8 securely realizes functionality $\mathcal{F}^1_{\rm CPRF}$ with respect to ${\rm PRF}_1$ in the presence of malicious adversaries in the $\{\mathcal{F}_{\rm 2DL}, \mathcal{F}_{\rm Eq-Exp}, \mathcal{F}_{\rm MULT}, \pi_{\rm DDH}, \mathcal{F}_{\rm EQ}\}$-hybrid model.*

Intuitively speaking, the security proof follows due to the ZK proofs that force the parties to behave honestly, and is a direct extension of the security proof of Protocol 3. A protocol based on ${\sf PRF}_2$ can be constructed similarly to the idea presented in Section 3.3.

5.1.2 The Complete Protocol

Next, we describe our set-intersection protocol using committed oblivious PRF.

Protocol 9 (Protocol $\pi_\cap$ with malicious security from committed oblivious PRF.)

- **Input:** *Party P_0 is given a set X of size m_X. Party P_1 is given a set Y of size m_Y. Both parties are given a security parameter 1^n.*
- **The protocol:**
  1. **Distributed key generation.** *P_0 and P_1 run protocol $\pi_{\rm KeyGen}(1^n, 1^n)$ in order to generate additive El Gamal public key ${\rm PK} = \langle \mathbb{G}, p, g, h\rangle$ where the corresponding shares of the secret key ${\rm SK}$ are $({\rm SK}_0, {\rm SK}_1)$.*
  2. **Input commitment and PRF key generation.** *P_0 sends encryptions of its input X under ${\rm PK}$; denote this set of ciphertexts by $C = (c_1, \ldots c_{m_X})$.*
     P_0 invokes $(K, param) \leftarrow {\sf KeyGen}(1^n, d = \log(m_X + m_Y))$ where $param$ includes a group description $\mathbb{G}$ of prime order p and a generator g, and sends P_1 param, and a ciphertext ${\sf Enc}_{\sf PK}(g^K; R)$.
     P_1 picks a new key $(K', param) \leftarrow {\sf KeyGen}(1^n, d = \log(m_X + m_Y))$ and sends it to P_0. The parties then compute the encryption $c_{\rm KEY}$ of $g^{\widetilde{K}} = g^{KK'}$, relying on the homomorphic property of El Gamal.
  3. **PRF evaluations on X.** *The parties call functionality $\mathcal{F}^0_{\rm CPRF}$ where P_0 enters the set X, key $\widetilde{K}$ and randomness R and P_1 enters $C, c_{\rm KEY}$ and ${\rm PK}$. Denote by ${\sf PRF}_X = \{{\sf PRF}'_{\widetilde{K}}(x)\}_{x\in X}$ the output of P_1 from this ideal call only if C is a vector of ciphertexts that encrypts X and $c_{\rm KEY}$ is a commitment of $\widetilde{K}$, where verification is computed using randomness R.*
  4. **Oblivious PRF evaluations on Y.** *The parties call functionality $\mathcal{F}^1_{\rm CPRF}$ where P_0 enters the key $\widetilde{K}$ and randomness R and P_1 enters the commitment $c_{\rm KEY}$ and the set Y. Denote by ${\sf PRF}_Y = \{f_y\}_{y\in Y}$ the output of P_1 from this ideal call only if $c_{\rm KEY}$ is a commitment of $\widetilde{K}$ where verification is computed using randomness R.*
     P_1 outputs all $y \in Y$ for which $f_y \in {\sf PRF}_X$.

Theorem 5.5 *Assume $\Pi = (\pi_{\rm KeyGen}, {\sf Enc}, \pi_{\rm DEC})$ is as in Section 3.3 and ${\sf PRF}'_K(\cdot)$ is a PRF according to Theorem 5.1, then Protocol 9 securely realizes functionality $\mathcal{F}_\cap$ in the presence of malicious adversaries in the $\{\mathcal{F}^0_{\rm CPRF}, \mathcal{F}^1_{\rm CPRF}\}$-hybrid model.*

Proof: We prove security for each corruption case separately. We assume that the simulator is given m_X and m_Y as part of its auxiliary input.

P_0 **is corrupted.** Let $\mathcal{A}$ be a PPT adversary corrupting party P_0, we design a PPT simulator $\mathcal{S}$ that generates the view of $\mathcal{A}$ as follows.

1. Given $(1^n, X, z)$, $\mathcal{S}$ engages in an execution of $\pi_{\text{KeyGen}}(1^n, 1^n)$ with $\mathcal{A}$. Denote the outcome by PK.
2. $\mathcal{S}$ receives the encryptions $C = (c_1, \dots c_{m_X})$ of $\mathcal{A}$'s input.
3. Upon receiving from $\mathcal{A}$ its commitment for the PRF key K, $\mathcal{S}$ picks a new key share $K' \leftarrow \mathsf{KeyGen}(1^n, d = \log(m_X + m_Y))$ and sends it to $\mathcal{A}$.
4. $\mathcal{S}$ extracts the adversary's input X' from the input to the ideal call $\mathcal{F}^0_{\text{CPRF}}$ and checks for consistency with the set C. If so, then $\mathcal{S}$ sends X' to the trusted party and completes the execution as would the honest P_1 do on an arbitrary set.
5. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

Claim 5.1 *For any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_\cap, \mathcal{S}(z)}(n, (X, Y))\}_{n \in \mathbb{N}} \overset{\text{s}}{\approx} \{\mathbf{HYBRID}_{\pi_\cap^{\{\mathcal{F}_{\text{DL}}, \mathcal{F}^0_{\text{CPRF}}, \mathcal{F}^1_{\text{CPRF}}\}}, \mathcal{A}(z)}(n, (X, Y))\}_{n \in \mathbb{N}}.$$

It is simple to verify that the adversary's view is identical in both executions, as the adversary does not receive any message from P_1 that depends on Y. An important observation here is that the probability of the event for which there exists $y \in Y$ such that $y \notin X'$ and yet $\mathsf{PRF}_{\widetilde{K}}(y) \in \mathsf{PRF}_{X'}$ is negligible, since the key $\widetilde{K}$ is picked independently of the set X'. This argument follows similarly to the proof in [HL10a] and implies that P_1's output in both executions is identical condition that the above event does not occur.

P_1 **is corrupted.** Let $\mathcal{A}$ be a PPT adversary corrupting party P_1, we design a PPT simulator $\mathcal{S}$ that generates the view of $\mathcal{A}$ as follows.

1. Given $(1^n, Y, z)$, $\mathcal{S}$ engages in an execution of $\pi_{\text{KeyGen}}(1^n, 1^n)$ with $\mathcal{A}$. Denote the outcome by PK.
2. $\mathcal{S}$ picks a set of m_X arbitrary elements $X_\mathcal{S}$ from $\mathbb{Z}_p$ and sends the encryptions $C = (c_1, \dots c_{m_X})$ of this set to $\mathcal{A}$.
3. $\mathcal{S}$ picks a PRF key share $K \leftarrow \mathsf{KeyGen}(1^n, d = \log(m_X + m_Y))$ and sends its encryption to $\mathcal{A}$ using PK. Upon receiving $\mathcal{A}$'s key share K' the simulator sets the combined key by $\widetilde{K} = KK'$.
4. $\mathcal{S}$ then emulates the ideal call $\mathcal{F}^0_{\text{CPRF}}$ and hands the adversary a random set U of size m_X of proper length.
5. Finally, the simulator extracts the adversary's input Y' to the ideal call $\mathcal{F}^1_{\text{CPRF}}$ and sends this set to the trusted party, receiving back $Z = X \cap Y'$. The simulator completes the execution as follows. For each element $y' \in Y' \cap Z$ it programs the ideal answer of $\mathcal{F}^1_{\text{CPRF}}$ to be $r \in U$ where r is picked from the remaining elements from the set U that were not picked thus far. Otherwise, the simulator returns a fresh random element from $\mathbb{Z}_p$.
6. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

Next we prove the following claim.

Claim 5.2 *For any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_\cap,\mathcal{S}(z)}(n,(X,Y))\}_{n\in\mathbb{N}} \overset{c}{\approx} \{\mathbf{HYBRID}_{\pi_\cap^{\{\mathcal{F}_{\rm DL},\mathcal{F}^0_{\rm CPRF},\mathcal{F}^1_{\rm CPRF}\}},\mathcal{A}(z)}(n,(X,Y))\}_{n\in\mathbb{N}}.$$

Proof: The proof follows by the IND-CPA security of the El Gamal PKE and the security of the PRF. That is, the simulated view is different from the hybrid view relative to the encrypted input of P_0 and the fact that the simulator uses a random function to evaluate the sets $X'_{\mathcal{S}}$ and Y'. Our proof follows by defining hybrid games and proving indistinguishability from both the simulated and the hybrid executions.

Hybrid $\mathbf{Hyb}_1$. Consider a hybrid game $\mathbf{Hyb}_1$ where there is no trusted party and the simulator $\mathcal{S}_{\mathbf{Hyb}_1}$ uses P_0's real input X for the PRF evaluations (namely, it uses X as an input to functionality $\mathcal{F}^0_{\rm PRF}$) but completes the simulation as in the original simulation (namely, sends encryptions of fake input $X_{\mathcal{S}}$ and fake PRF key). More formally, we mark in bold the underlying differences from the simulation.

1. Given $(1^n, \mathbf{X}, Y, z)$, $\mathcal{S}_{\mathbf{Hyb}_1}$ engages in an execution of $\pi_{\rm KeyGen}(1^n, 1^n)$ with $\mathcal{A}$. Denote the outcome by PK.
2. $\mathcal{S}$ picks a set of m_X arbitrary elements $X_{\mathcal{S}}$ from $\mathbb{Z}_p$ and sends the encryptions $C = (c_1, \ldots c_{m_X})$ of this set to $\mathcal{A}$.
3. $\mathcal{S}$ picks a PRF key share $K \leftarrow \mathsf{KeyGen}(1^n, d = \log(m_X + m_Y))$ and sends its encryption to $\mathcal{A}$ using PK. Upon receiving $\mathcal{A}$'s key share K' the simulator sets the combined key by $\widetilde{K} = KK'$.
4. $\mathcal{S}_{\mathbf{Hyb}_1}$ **emulates the ideal call $\mathcal{F}^{\mathbf{0}}_{\rm CPRF}$ on the set X and key KK$'$**, and hands the adversary a random set U of size m_X of proper length.
5. Finally, the simulator extracts the adversary's input Y' to the ideal call $\mathcal{F}^1_{\rm CPRF}$ and computes $Z = X \cap Y'$. It then completes the execution as follows. For each element $y' \in Y' \cap Z$ it programs the ideal answer of $\mathcal{F}^1_{\rm CPRF}$ to be $r \in U$ where r is picked from the remaining elements from the set U that were not picked thus far. Otherwise, the simulator returns a fresh random element from $\mathbb{Z}_p$.

SubClaim 5.3 *For any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_\cap,\mathcal{S}(z)}(n,(X,Y))\}_{n\in\mathbb{N}} \equiv \{\mathbf{Hyb}^1_{\pi_\cap^{\{\mathcal{F}_{\rm DL},\mathcal{F}^0_{\rm CPRF},\mathcal{F}^1_{\rm CPRF}\}},\mathcal{A}(z)}(n,(X,Y))\}_{n\in\mathbb{N}}.$$

Proof: It is easy to verify that the distributions are identical as in both executions the simulator uses a random function for which its outcomes are independent of the inputs to the function. □

Hybrid $\mathbf{Hyb}_2$. Next, consider a hybrid game $\mathbf{Hyb}_2$ where the simulator $\mathcal{S}_{\mathbf{Hyb}_2}$ uses a PRF instead of a random function when emulating functionalities $\mathcal{F}^0_{\rm CPRF}$ and $\mathcal{F}^1_{\rm CPRF}$. Indistinguishability between game $\mathbf{Hyb}_1$ and game $\mathbf{Hyb}_2$ follows by a reduction to the pseudorandomness of the PRF, where the distinguisher can simply plug-in the evaluations obtained from its oracle in the messages to the adversary. More formally, we mark in bold the underlying differences from the simulation.

1. Given $(1^n, X, Y, z)$, $\mathcal{S}_{\mathbf{Hyb}_2}$ engages in an execution of $\pi_{\rm KeyGen}(1^n, 1^n)$ with $\mathcal{A}$. Denote the outcome by PK.
2. $\mathcal{S}$ picks a set of m_X arbitrary elements $X_{\mathcal{S}}$ from $\mathbb{Z}_p$ and sends the encryptions $C = (c_1, \ldots c_{m_X})$ of this set to $\mathcal{A}$.

3. $\mathcal{S}$ picks a PRF key share $K \leftarrow \mathsf{KeyGen}(1^n, d = \log(m_X + m_Y))$ and sends its encryption to $\mathcal{A}$ using PK. Upon receiving $\mathcal{A}$'s key share K' the simulator sets the combined key by $\widetilde{K} = KK'$.

4. $\mathcal{S}_{\mathbf{Hyb}_2}$ emulates the ideal call $\mathcal{F}^0_{\mathrm{CPRF}}$ on the set X **and key** $\widetilde{\mathbf{K}} = \mathbf{KK'}$, and hands the adversary the set $\mathsf{PRF}_X = \{\mathsf{PRF}'_{\widetilde{K}}(x)\}_{x \in X}$.

5. Finally, the simulator extracts the adversary's input Y' to the ideal call $\mathcal{F}^1_{\mathrm{CPRF}}$ **and hands the adversary the set** $\mathsf{PRF}_{\mathbf{Y}'} = \{\mathsf{PRF}'_{\widetilde{\mathbf{K}}}(\mathbf{y}')\}_{\mathbf{y}' \in \mathbf{Y}'}$.

SubClaim 5.4 *For any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\mathbf{Hyb}^1_{\pi_\cap^{\{\mathcal{F}_{\mathrm{DL}}, \mathcal{F}^0_{\mathrm{CPRF}}, \mathcal{F}^1_{\mathrm{CPRF}}\}}, \mathcal{A}(z)}(n, (X, Y))\}_{n \in \mathbb{N}} \equiv \{\mathbf{Hyb}^2_{\pi_\cap^{\{\mathcal{F}_{\mathrm{DL}}, \mathcal{F}^0_{\mathrm{CPRF}}, \mathcal{F}^1_{\mathrm{CPRF}}\}}, \mathcal{A}(z)}(n, (X, Y))\}_{n \in \mathbb{N}}.$$

Proof: Assume by contradiction the existence of an adversary $\mathcal{A}$ and a distinguisher $D_\cap$ that distinguishes $\mathcal{A}$'s views for infinitely many n's, construct a distinguisher D_F that distinguishes F from a truly random function f that maps $\{0,1\}^m$ to $\mathbb{G}$, for infinitely many n's. Namely, upon given access to an oracle $\mathcal{O}$ that implements either F or f, and auxiliary input $(1^n, X, Y, z)$, D_F invokes $\mathcal{A}$ on $1^n, Y, z$. It then computes the messages of Protocol 9 as follows. It first invokes its oracle on the set X, let f_X denote the set of the oracle's responses. D_F then completes the execution as in $\mathbf{Hyb}_2$ with the exception that it uses f_X as the input given to $\mathcal{A}$ from $\mathcal{F}^0_{\mathrm{CPRF}}$. Furthermore, it uses its oracle to evaluate the set Y' submitted by the adversary to $\mathcal{F}^1_{\mathrm{CPRF}}$. Finally, D_F invokes $D_\cap$ on $\mathcal{A}$'s view and outputs whatever $D_\cap$ does. Note that if oracle $\mathcal{O}$ implements function F, then $\mathcal{A}$'s view distributes as in the hybrid game. On the other hand, if $\mathcal{O}$ implements f then $\mathcal{A}$'s view distributes as in the simulation (this is because the distribution on the PRF values is independent of the actual inputs). Therefore, any gap in distinguishing these two views is immediately translated into a gap distinguishing F from f. $\square$

Indistinguishability between game $\mathbf{Hyb}_2$ and the hybrid execution follows by a reduction to the IND-CPA security of El Gamal since the simulator never uses the secret key of the encryption scheme and the only difference between the executions is with respect to the set of ciphertexts C.

SubClaim 5.5 *or any tuple of inputs (X, Y) and auxiliary input z,*

$$\{\mathbf{Hyb}^2_{\pi_\cap^{\{\mathcal{F}_{\mathrm{DL}}, \mathcal{F}^0_{\mathrm{CPRF}}, \mathcal{F}^1_{\mathrm{CPRF}}\}}}(n, (X, Y))\}_{n \in \mathbb{N}} \overset{c}{\approx} \{\mathbf{HYBRID}_{\pi_\cap^{\{\mathcal{F}_{\mathrm{DL}}, \mathcal{F}^0_{\mathrm{CPRF}}, \mathcal{F}^1_{\mathrm{CPRF}}\}}, \mathcal{A}(z)}(n, (X, Y))\}_{n \in \mathbb{N}}.$$

Proof: Assume by contradiction the existence of an adversary $\mathcal{A}$ and a distinguisher $D_\cap$ that distinguishes $\mathcal{A}$'s views for infinitely many n's, construct a distinguisher D_E that breaks the security of El Gamal for infinitely many n's. Namely, upon given access to an encryption oracle $\mathcal{O}$, and auxiliary input $(1^n, X, Y, z)$, D_E invokes $\mathcal{A}$ on $1^n, Y, z$. It then computes the messages of Protocol 9 as follows. It first invokes its oracle on the sets (X, g^K) and $(X_\mathcal{S}, g^U)$ where $K \leftarrow \mathsf{KeyGen}(1^n, d = \log(m_X + m_Y))$ and $U \leftarrow \mathbb{Z}_p$, let E denote the set of the oracle's responses. D_E then sends $\mathcal{A}$ the ciphertexts within E that correspond to either X or $X_\mathcal{S}$, as part of its input commitments, and uses the remaining ciphertext for the PRF key generation. D_E completes the execution as in $\mathbf{Hyb}_2$ with the exception that it uses KK' as the PRF key given to $\mathcal{A}$ from $\mathcal{F}^0_{\mathrm{CPRF}}$. Finally, D_F invokes $D_\cap$ on $\mathcal{A}$'s view and outputs whatever $D_\cap$ does. Note that if oracle $\mathcal{O}$ encrypts the set (X, g^K), then $\mathcal{A}$'s view distributes as in the hybrid game (where the correct input set X and PRF key are encrypted). On the other hand, if $\mathcal{O}$ encrypts $(X_\mathcal{S}, g^U)$ then $\mathcal{A}$'s view distributes as in the simulation (this is because the adversary sees the encryptions of the fake input set and a fake PRF key that is never used). Therefore, any gap in distinguishing these two views is immediately translated into a gap distinguishing F from f. $\square$

This concludes the proof. $\square$ $\blacksquare$

Efficiency. The overhead of protocol 9 depends on the implementations of $\mathcal{F}^0_{\mathrm{CPRF}}$ and $\mathcal{F}^1_{\mathrm{CPRF}}$ discussed above. Our protocol obtains $O(m_X + m_Y)$ communication and computation overheads under the strong-DDH assumption and $O((m_X + m_Y)\log(m_X + m_Y))$ overheads under the DDH assumption, where the former analysis matches the [JL09] analysis (such that both constructions rely on dynamic assumptions).

5.2 Search Functionalities

In search functionalities a receiver searches in a sender's database, retrieving the appropriate record(s) according to some search query. The database for search functionalities can be described by pairs of queries/records $\{(q_i, T_i)\}_i$ such that the answer to a query q_i is a record T_i.[13] In a private setting we need to ensure that nothing beyond these records leaks to the receiver, while the sender does not learn anything about the receiver's search queries. Committed oblivious PRF evaluation is a useful tool for securely implementing various search functionalities [FIPR05]. First, in the setup phase the database is encoded and handed to the receiver. That is, for each query q_i the sender defines the pair $(\mathsf{PRF}_K(q_i\|1), \mathsf{PRF}_K(q_i\|2) \oplus T_i)$. Next, in the query phase the parties run a committed oblivious PRF evaluation protocol twice such that the sender inputs K and the receiver inputs a query q. The receiver's output are the values $\mathsf{PRF}_K(q\|1)$ and $\mathsf{PRF}_K(q\|2)$, where the first outcome is used to find the encrypted record while the second outcome is used to extract the record. (Alternative implementations involve a single invocation of PRF by splitting $\mathsf{PRF}_K(q)$ into two parts). Examples for such functionalities are keyword search, oblivious transfer with adaptive queries and pattern matching (and all its variants). The functionality of committed oblivious PRF is important in this context since the sender must be enforced to use the same PRF key.

References

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

[ACT10] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: Size-hiding private set intersection. *IACR Cryptology ePrint Archive*, 2010:220, 2010.

[ADDV12] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled cloud storage. *IACR Cryptology ePrint Archive*, 2012:511, 2012.

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

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131, 2011.

[BM01] Andrei Z. Broder and Michael Mitzenmacher. Using multiple hash functions to improve IP lookups. In *INFOCOM*, pages 1454–1463, 2001.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

[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.

[13]This may not be the most concise description of the database but it is the simplest. In particular, it will do for our purposes.

[CL05] Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. *Theor. Comput. Sci.*, 341(1-3):39–54, 2005.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.

[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 efcient and scalable protocol. *IACR Cryptology ePrint Archive*, 2013:515, 2013.

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6):371–385, 2010.

[DKL$^+$13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 1–18, 2013.

[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.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, pages 416–431, 2005.

[FHV13] Sebastian Faust, Carmit Hazay, and Daniele Venturi. Outsourced pattern matching. In *ICALP*, 2013.

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

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

[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.

[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.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[Haz15] Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 90–120, 2015.

[HL09] Carmit Hazay and Yehuda Lindell. Efficient oblivious polynomial evaluation with simulation-based security. *IACR Cryptology ePrint Archive*, 2009:459, 2009.

[HL10a] 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.

[HL10b] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols – Techniques and Constructions*. Springer-Verlag, 2010.

[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.

[KMW08] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. In *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, pages 611–622, 2008.

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

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO (2)*, pages 1–17, 2013.

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

[LP02] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, pages 329–346, 2011.

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

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. Lego for two-party secure computation. In *TCC*, pages 368–386, 2009.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.

[NP06] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, pages 458–467, 1997.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, pages 31–53, 1992.

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

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX*, pages 515–530, 2015.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 797–812, 2014.

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

[ST04] Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In *ASIACRYPT*, pages 119–136, 2004.

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

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

[ZB05] Huafei Zhu and Feng Bao. Augmented oblivious polynomial evaluation protocol and its applications. In *ESORICS*, pages 222–230, 2005.

A Secure Two-Party Computation

We briefly present the standard definition for secure multiparty computation and refer to [Gol04, Chapter 7] for more details and motivating discussions. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a *functionality* and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-vector is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings where P_1 receives $f_1(x, y)$ and P_2 receives $f_2(x, y)$. We use the notation $(x, y) \mapsto (f_1(x, y), f_2(x, y))$ to describe a functionality.

We prove the security of our protocols in the settings of *semi-honest* and *malicious* computationally bounded adversaries. Security is analyzed by comparing what an adversary can do in a *real* protocol execution to what it can do in an *ideal* scenario. In the ideal scenario, the computation involves an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Informally, the protocol is secure if any adversary interacting in the real protocol (i.e., where no trusted third party exists) can do no more harm than what it could do in the ideal scenario.

A.1 The Semi-Honest Setting

In this model the adversary controls one of the parties and follows the protocol specification. However, it may try to learn more information than allowed by looking at the transcript of messages that it received and its internal state. Let $f = (f_1, f_2)$ be a two-party functionality and let π be a two-party protocol for computing f. The *view* of the first party in an execution of π on inputs (x, y) is

$$\mathbf{View}_{\pi,1}(x, y) = (x, r_1, m_1, \ldots, m_t),$$

where r_1 is the content of the first party's internal random tape, and m_i represents the ith message that it received. The *output* of the first party in an execution of π on (x, y) is denoted $\mathbf{Output}_{\pi,1}(x, y)$ and can be computed from $\mathbf{View}_{\pi,1}(x, y)$. Similarly, $\mathbf{View}_{\pi,2}(x, y)(y, r_2, m_1, \ldots, m_t)$ where r_2 is second party's randomness and m_i is the ith message it received. The output of the second party can be computed from her view and is denoted $\mathbf{Output}_{\pi,2}(x, y)$.

Definition A.1 *Let f and π be as above. Protocol π is said to securely compute f in the presence of* semi-honest *adversaries if there exist probabilistic polynomial-time algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ such that*

$$(\mathcal{S}_1(x, f_1(n, x, y)), f_2(n, x, y))_{n\in\mathbb{N},x,y\in\{0,1\}^*} \overset{c}{\approx} \{(\mathbf{View}_{\pi,1}(n, x, y), \mathbf{Output}_{\pi,2}(n, x, y))\}_{n\in\mathbb{N},x,y\in\{0,1\}^*}$$

$$(f_1(n,x,y), \mathcal{S}_2(y, f_2(n,x,y)))_{n\in\mathbb{N},x,y\in\{0,1\}^*} \overset{\text{c}}{\approx} \{(\mathbf{Output}_{\pi,1}(n,x,y), (\mathbf{View}_{\pi,2}(n,x,y)))\}_{n\in\mathbb{N},x,y\in\{0,1\}^*}$$

where n is the security parameter.

A.2 The Malicious Setting

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

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

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

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

Definition A.2 *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [2]$,*

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

where n is the security parameter.

The $\mathcal{F}$-hybrid model. In order to construct some of our protocols, we will use secure two-party protocols as subprotocols. The standard way of doing this is to work in a "*hybrid model*" where parties both interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, when constructing a protocol π that uses a subprotocol for securely computing some functionality $\mathcal{F}$, we consider the case that the parties run π and use "ideal calls" to a trusted party for computing $\mathcal{F}$. Upon receiving the inputs from the parties, the trusted party computes $\mathcal{F}$ and sends all parties their output. Then, after receiving these outputs back from the trusted party the protocol π continues.

Let $\mathcal{F}$ be a functionality and let π be a two-party protocol that uses ideal calls to a trusted party computing $\mathcal{F}$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine. Then, the *$\mathcal{F}$-hybrid execution of* π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\mathbf{HYBRID}_{\pi^{\mathcal{F}},\mathcal{A}(z)}(n, x, y)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing $\mathcal{F}$. By the composition theorem of [Can00] any protocol that securely implements $\mathcal{F}$ can replace the ideal calls to $\mathcal{F}$.