Efficient Set Operations in the Presence of Malicious Adversaries

Carmit Hazay* Kobbi Nissim†

December 2, 2009

Abstract

We revisit the problem of constructing efficient secure two-party protocols for the problems of set-intersection and set-union, focusing on the model of malicious parties. Our main results are constant-round protocols that exhibit linear communication and a (practically) linear number of exponentiations with simulation based security. In the heart of these constructions is a technique based on a combination of a perfectly hiding commitment and an oblivious pseudorandom function evaluation protocol. Our protocols readily transform into protocols that are UC-secure, and we discuss how to perform these transformations.

Keywords: Secure two-party computation, Simulation-based security, Set-intersection, Set-union, Oblivious pseudorandom function evaluation.

1 Introduction

Secure function evaluation (SFE) allows two distrusting parties to jointly compute a function of their respective inputs as if the computation is executed in an ideal setting where the parties send inputs to a trusted party that performs the computation and returns its result. Starting with the work of [46, 27, 11, 4], it is by now well known that (in various settings, and considering semi-honest and malicious adversaries) any polynomial-time computation can be generically compiled into a secure function evaluation protocol with polynomial complexity. However, more often than not, the resulting protocols are inefficient for practical uses and hence attention was given to constructing efficient protocols for specific functions. This approach that proved quite successful for the semi-honest setting (see, e.g., [35, 20, 37, 1, 24, 33, 5, 39, 32, 36]), while the malicious setting remained, at large, elusive (a notable exception is [1]).

We focus on the secure computation of basic set operations (intersection and union) where the parties P_1, P_2, holding input sets X, Y, respectively, wish to compute $X \cap Y$ or $X \cup Y$. These problems have been widely looked at by researchers in the last few years [24, 33, 28, 6, 30, 8, 18], mainly due their potential applications for dating services, datamining, recommendation systems, law enforcement, etc. Note that a general protocol for these problems (e.g., [29, 31]), uses a circuit of size $\Omega(|X| \cdot |Y|)$. Therefore, our main goal in this work is to come up with protocols for set-intersection and set-union that are fully secured in the malicious setting and are of better complexity to those known (including the above general results).

We begin by briefly surveying the current state of affairs with respect to two-party secure computation of these functions for papers that relate to our work.

*Dept. of Computer Science and Applied Mathematics, Weizmann Institute and IDC. carmit.hazay@weizmann.ac.il. Research was supported by an Eshkol scholarship.

†Dept. of Computer Science, Ben-Gurion University and Microsoft ILDC. kobbi@cs.bgu.ac.il. Research partly supported by the Israel Science Foundation (grant No. 860/06).

- Freedman, Nissim and Pinkas studied set intersection in [24]. They represent a set by a polynomial that zeros on the element of the set. Their construction for the semi-honest setting utilizes oblivious polynomial evaluation and a balanced allocation scheme and exhibits linear communication (counting field elements) and (almost) linear computation (counting modular exponentiations). We give a detailed account of this protocol in Section 3.

  Freedman et al. also present variants of the above protocol, for the case where one of the parties is malicious and the other is semi-honest. In both cases, generic zero-knowledge proofs of adherence to the protocol are avoided to gain in efficiency. The protocol for malicious P_1 (*client* in [24]) and semi-honest P_2 (*server*) utilizes a cut-and-choose strategy and hence communication is inflated by a statistical security parameter. The protocol for malicious P_2 and semi-honest P_1 is in the random oracle model. A protocol that is secure in the fully malicious setup, that combines both techniques, is sketched in Section 3.1.

- Kissner and Song [33] used polynomials to represent multi-sets. Letting the roots of $Q_X(\cdot)$ and $Q_Y(\cdot)$ coincide with elements of the multi-sets X and Y, Kissner and Song observed that if $r(\cdot), s(\cdot)$ are polynomials chosen at random then the roots of $r(\cdot) \cdot Q_X(\cdot) + s(\cdot) \cdot Q_Y(\cdot)$ coincide with high probability with the multi-set $X \cap Y$. This beautiful observation yields a set-intersection protocol for the semi-honest case, where the parties use an additively homomorphic encryption scheme (the Paillier scheme is suggested in [33]) to perform the polynomial multiplication, introducing quadratic computation costs in the set sizes. For the security of the protocol, it is crucial that no party should be able to decrypt on her own. Hence, the secret key should be shared and joint decryption should be deployed. Assuming a trusted setup for the encryption scheme, the communication costs for the two-party case are as in the protocol for semi-honest parties of [24].

  For malicious parties [33] introduced generic zero-knowledge proofs for proving adherence to the prescribed protocol (e.g., zero-knowledge proofs of knowledge for the multiplication of the encrypted $Q_x(\cdot)$ with a randomly selected $r(\cdot)$). While this change seems to be of dire consequences to the protocol efficiency, the analysis in [33] ignores its effects. Furthermore, the costs of setting the shared key for the Paillier scheme are ignored in the analysis. To the best of our knowledge, there are currently no efficient techniques for generating the shared Paillier keys, which do not incorporate an external trusted dealer (the latter schemes include [21, 22] referenced in [33]).

  In addition to that, Kissner and Song presented a protocol for the threshold set-union problem, where only the elements that appear in the combined inputs more than t times are learnt by the parties. Their protocol employs the same technique of polynomial multiplication and thus introduces quadratic computation costs as above.

- Hazay and Lindell [28] revisited secure set intersection, with the aim of achieving efficient protocols in presence of a more realistic adversarial behavior than in the benign semi-honest model, and under standard cryptographic assumptions. Two protocols were presented, one achieves security in the presence of malicious adversaries with one-sided simulatability, the other is secure in the presence of covert adversaries [2]. The main tool used in these protocols is a secure implementation of oblivious pseudorandom function evaluation.

  Having P_1, P_2 hold sets of sizes m_X, m_Y respectively, both protocols in [28] are constant round, and incur the communication of $O(m_X \cdot p(n) + m_Y)$ group elements and the computation of $O(m_X \cdot p(n) + m_Y)$ modular exponentiations, where set elements are taken from $\{0,1\}^{p(n)}$.

  We note that the protocols in [28] can be made secure in the malicious setup, e.g., by introducing a secure key selection step for the oblivious PRF and by adding generic zero-knowledge proofs to

show correctness at each step. While this would preserve the complexity of these protocols asymptotically (in m_X, m_Y), the introduction of generic zero-knowledge proofs would probably make them inefficient for practical use.

- Recently, Jarecki, and Liu [30] presented a very efficient protocol for computing a pseudorandom function with a committed key (informally, this means that the same key is used in all invocations), and showed that it yields an efficient set-intersection protocol. The main restriction of this construction is that the input domain size of the pseudorandom function should be polynomial in the security parameter (curiously, the proof of security for the set-intersection protocol makes use of the ability to exhaustively search over the input domain, so removing the restriction on the input domain of the pseudorandom function does not immediately yield a set-intersection protocol for a super-polynomial domain).

1.1 Our Contributions

Our main contributions are efficient set-intersection and set-union protocols that are secure in the setup of malicious parties. Our constructions are in the standard model, and are based on standard cryptographic assumptions (in particular, no random oracle or a trusted setup).

Naturally, our main goal is to come up with protocols that exhibit low asymptotic communication and computation costs. Noting that asymptotic complexity does not reveal everything about a protocol's efficiency or practicality, we avoid using generic zero-knowledge proofs of adherence to the prescribed protocols, even when they involve relatively short statements, and costly set up commutations that make the efficient only for very large inputs.

Preventing the Players from Deviating from the Protocol: We inherit the oblivious polynomial evaluation and balanced allocation techniques used in [24]. On top of these we introduce an efficient zero-knowledge proof that P_1 uses to show that her encrypted polynomials were correctly produced (unlike in [24], our proof does not use a cut-and-choose strategy), and a technique preventing player P_2 from deviating meaningfully from the protocol. This technique combines a perfectly hiding commitment scheme with an oblivious pseudorandom function evaluation protocol. In some sense, this construction replaces the random oracle used in [24] in the case of a malicious sender, but this 'replacement' is only in a very weak sense: In our construction P_2 holds the key for the pseudorandom function, and hence the function does not look random to P_2. Furthermore, P_2 does not need to invoke the oblivious pseudorandom evaluation protocol to compute it. The consequence is that, unlike with the simulator for the protocol in the random oracle model that can easily monitor all invocations of the oracle, our simulator cannot extract P_2's input to the pseudorandom function.

We note that the protocols of [28] also use an oblivious pseudorandom function evaluation primitive, where the the player analogous to P_2 knows the key for the function. Their usage of this primitive is, however, very unlike in our protocols. In the protocols of [28] the pseudorandom function is evaluated on the set elements P_2 holds, whereas in our protocols it is evaluated on a random payload. Furthermore, the protocols in [28] are designed for the covert adversary model and for the one-sided simulatability model, and hence a technique enabling the simulation of P_2 is not needed, whereas our constructions allow simulation of both parties.

Choosing the underlying encryption scheme: Our protocols make extensive use of a homomorphic encryption scheme, and would remain secure (with only small modifications) under a variety of choices. We chose to work with the El Gamal scheme (that is *multiplicatively* homomorphic) although it may seem

that the more natural choice is the Paillier scheme [41], that is additively homomorphic (indeed, our initial constructions were based on the Paillier scheme).

Using the Paillier scheme, a subtle problem emerges (this was overlooked, e.g., in [24]). Recall that for the Paillier scheme $pk = N, sk = \phi(N)$. Now, if P_1 knows sk when she constructs her polynomials, then she may construct a polynomial $Q(\cdot)$ such that $Q(y) \notin \mathbb{Z}_N^*$ for some specific 'target' value y. This would allow her to learn about P_2's input beyond the intended protocol output. A possible solution is that P_1, P_2 would first engage in a protocol to jointly generate pk and shares of sk, whereas P_1 would learn sk only *after* committing to her polynomials. This, however, introduces high key setup costs, and the result is a protocol that exhibits low *asymptotic* costs, but, because of its high setup costs, its efficiency is gained only for very large inputs.

Efficiency: Our protocols for set intersection and set union $\pi_\cap, \pi_\cup$ are constant round, work in the standard model and do not require a trusted setup. The underlying encryption scheme is El Gamal where the keys are selected by party P_1. Both protocols do not employ any generic zero-knowledge proof.

Assuming the protocol of [23] for the pseudorandom function evaluation we get that for sets $X, Y \subset \{0,1\}^{p(n)}$ of m_X, m_Y elements respectively, the costs of $\pi_\cap, \pi_\cup$ are of sending $O(m_X + m_Y \cdot p(n))$ group elements, and the computation of $O(m_X + m_Y \cdot (\log\log m_X + p(n)))$ modular exponentiations. Note that this is significantly better than $O(m_X \cdot m_Y)$.

A significant improvement can be achieved by using a more efficient pseudorandom function evaluation instead of using the function of [38] which requires a single oblivious transfer for every input bit. Furthermore, for set intersection, another significant improvement can be achieved if the size of the intersection $m_{X\cap Y}$ is allowed to be leaked (to P_2). The resulting protocol is of sending $O(m_X + m_{X\cap Y} \cdot p(n))$ and computing $O(m_X + m_Y \cdot \log\log m_X + m_{X\cap Y} \cdot p(n))$. When $m_{X\cap Y} \ll m_Y$ we get a protocol that is more efficient than that of [28].

UC security: Our protocols readily transform into the UC framework as all our simulators are straight-line in an hybrid model with access to some specific zero-knowledge proofs. We show how to modify our set intersection protocol to one that is secure in the UC framework (in the common reference string model).

2 Preliminaries

Throughout the paper, we denote the security parameter by n, and, although not explicitly specified, input lengths are always assumed to be bounded by some polynomial in n. A probabilistic machine is said to run in *polynomial-time* (PPT) if it runs in time that is polynomial in the security parameter n alone. A function $\mu(n)$ is *negligible* (in n) if for every polynomial $p(\cdot)$ there exists a value N such that $\mu(n) < \frac{1}{p(n)}$ for all $n > N$; i.e., $\mu(n) = n^{-\omega(1)}$.

Let $X = \{X(n,a)\}_{n\in N, a\in\{0,1\}^*}$ and $Y = \{Y(n,a)\}_{n\in N, a\in\{0,1\}^*}$ be distribution ensembles (over strings of length polynomial in n). We say that X and Y are *computationally indistinguishable*, denoted $X \stackrel{c}{\equiv} Y$, if for every polynomial non-uniform distinguisher D there exists a negligible $\mu(\cdot)$ such that

$$\Big|\Pr[D(X(n,a)) = 1] - \Pr[D(Y(n,a)) = 1]\Big| < \mu(n) \qquad \text{for every } n \in N \text{ and } a \in \{0,1\}^*.$$

2.1 Secure Two-Party Computation – Definitions

We briefly present the standard definition for secure multiparty computation and refer to [25, Chapter 7] for more details and motivating discussion.

Two-party computation. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a *functionality* and denote it $f = (f_1, f_2) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. 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 the outputs of P_1, P_2 are $f_1(x, y), f_2(x, y)$ respectively. We use the notation $(x, y) \mapsto (f_1(x, y), f_2(x, y))$ to describe a functionality. For example, the Oblivious Transfer functionality is written $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where (x_0, x_1) is the first party's input, σ is the second party's input, and λ denotes the empty string (meaning that the first party has no output). A special case for a two-party functionality is that of zero-knowledge proof of knowledge for a relation $\mathcal{R}_{\text{ZK}}$. This functionality is defined as:

$$(x, (x, w)) \mapsto \begin{cases} (1, \lambda) & \text{if } \mathcal{R}_{\text{ZK}}(x, w) = 1 \\ (\perp, \lambda) & \text{otherwise} \end{cases}$$

Security of protocols. We prove the security of our protocols in the setting of *malicious adversaries*, that may arbitrarily deviate from the specified protocol. 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. We consider the *static* setting where the adversary is only able to corrupt a party at the outset of the protocol. There are technical issues that arise, such as that it may be impossible to achieve fairness or guaranteed output delivery. E.g., it is possible for the an adversarial party to prevent an honest party from receiving outputs.

Execution in the ideal model. In an ideal execution, the parties send 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 output of the corrupted parties to the adversary, and the adversary then decides whether the honest parties receive their (correct) outputs or an abort symbol $\perp$. Let f be a two-party functionality where $f = (f_1, f_2)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subseteq [2]$ be the set of corrupted parties (either P_1 is corrupted or P_2 is corrupted or neither). Then, the *ideal execution of* f on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{IDEAL}_{f,\mathcal{A}(z),I}(x, y, n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. In the real model there is no trusted third party and the parties directly interact with each other. The adversary $\mathcal{A}$ controls the corrupted parties and hence sends all messages in their place. The adversary is not bound to sending messages according to the protocol, and may follow an arbitrary polynomial-time strategy. An honest party follows the instructions prescribed in the protocol π.

Let f be as above and let π be a two-party protocol for computing f. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let $I \subseteq [2]$ be the set of corrupted parties. Then, the *real execution of* π on inputs (x, y), auxiliary input z to $\mathcal{A}$, and security parameter n, denoted $\text{REAL}_{\pi,\mathcal{A}(z),I}(x, y, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real execution models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure protocol emulates (in the real execution model) the ideal execution model in which a trusted party exists.

This notion is formulated by requiring the existence of adversaries in the ideal execution model that are able to simulate adversarial behavior in the real execution model.

Definition 2.1 *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subseteq [2]$,*

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

where $|x| = |y|$.

The f-hybrid model. In our constructions we will use secure two-party protocols as sub-protocols. A standard way of abstracting out the details of a sub-protocol computing a functionality f is to work in a *hybrid model* where the two parties directly interact with each other (as in the real model) as well as access a trusted implementation of f (as in the ideal model). In an execution of a protocol π that uses a sub-protocol for securely computing f, the parties run π and issue *ideal calls* to a trusted party for computing f instead of invoking the sub-protocol for f. These ideal calls are just instructions to send an input to the trusted party, which, upon receiving the inputs from the parties, computes f and sends each party its corresponding output. After receiving these outputs back from the trusted party, the protocol π continues. We stress that this is a *sequential* composition, i.e., the parties do not send messages in π between the time that they send input to the trusted party and the time that they receive back output. The trusted party may be used a number of times throughout the execution of π. However, each invocation of the functionality f is independent (i.e., the trusted party does not maintain any state between these calls). We call the regular messages of π that are sent amongst the parties *standard messages* and the messages that are sent between parties and the trusted party *ideal messages*.

Let f be a functionality and let π be a two-party protocol that uses ideal calls to a trusted party 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 *f-hybrid execution of* π on inputs (x,y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{HYBRID}^{f}_{\pi,\mathcal{A}(z),I}(x,y,n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing f.

Let f and π be as above, and let ρ be a protocol. Consider the real protocol π^ρ that is defined as follows. All standard messages of π are unchanged. When a party P_i is instructed to send an ideal message α_i to the trusted party, it begins a real execution of ρ with input α_i instead. When this execution of ρ concludes with output β_i, party P_i continues with π as if β_i was the output received by the trusted party (i.e. as if it were running in the f-hybrid model). Then, the composition theorem of [7] states that if ρ securely computes f, then the output distribution of a protocol π in a hybrid execution with f is computationally indistinguishable from the output distribution of the real protocol π^ρ. Thus, it suffices to analyze the security of π when using ideal calls to f; security of the real protocol π^ρ is derived via this composition theorem.

2.2 The El Gamal Encryption Scheme

The El Gamal encryption scheme operates on a cyclic group $\mathbb{G}$ of prime order q. We will work in the group $\mathbb{Z}^*_{q'}$ where $q' = 2q+1$ is prime, and set $\mathbb{G}$ to be the subgroup of $\mathbb{Z}_{q'}$ of quadratic residues modulo q' (note that membership in $\mathbb{G}$ can be easily checked). Let g denote a random generator in $\mathbb{G}$, then the public and secret keys are $\langle \mathbb{G}, q, g, h\rangle$ and $\langle \mathbb{G}, q, g, x\rangle$ where $x \leftarrow_R \mathbb{Z}_q$ and $h = g^x$. A message $m \in \mathbb{G}$ is encrypted by choosing $y \leftarrow_R \mathbb{Z}_q$ and the ciphertext is $\langle g^y, h^y \cdot m\rangle$. A ciphertext $c = \langle \alpha, \beta\rangle$ is decrypted as $m = \beta/\alpha^x$.

We use the property that given $y = \log_g \alpha$ one can reconstruct $m = \beta/h^y$ and hence a party encrypting m can prove knowledge of m by proving knowledge of y.

The semantic security of the El Gamal scheme follows from the hardness of decisional Diffie-Hellman (DDH) in $\mathbb{G}$. The El Gamal scheme is homomorphic relative to multiplication. I.e., if $\langle \alpha_1, \beta_1 \rangle$ encrypts m_1 and $\langle \alpha_2, \beta_2 \rangle$ encrypts m_2 then $\langle \alpha_1 \cdot \alpha_2, \beta_1 \cdot \beta_2 \rangle$ encrypts $m_1 m_2$.

2.3 Perfectly Hiding Commitment

We use a perfectly-hiding commitment scheme $(\mathsf{com}, \mathsf{dec})$ with a zero-knowledge proof of knowledge π_{COM} for the relation

$$\mathcal{R}_{\text{COM}} = \left\{ \Big(c, (r, m) \Big) \mid c = \mathsf{com}(m; r) \right\},$$

where $\mathsf{com}(m; r)$ denotes the commitment to a message m using random coins r. We instantiate $\mathsf{com}(\cdot;\cdot)$ with Pedersen's commitment scheme [42], using the same underlying group $\mathbb{G}$ used for the El Gamal scheme. I.e., let $q' = 2q + 1$ where q', q are primes and let g, h be generators of the subgroup $\mathbb{G}$ of quadratic residues modulo q'. A commitment to m is then defined as $\mathsf{com}(m; r) = g^m h^r$ where $r \leftarrow_R \mathbb{Z}_{q-1}$. The scheme is perfectly hiding as for every m, r, m' there exists a single r' such that $g^m h^r = g^{m'} h^{r'}$. The scheme is binding assuming hardness of computing $\log_g h$. However, given $\log_g h$, it is possible to de-commit any commitment c into any message $m \in \mathbb{Z}_q$. We instantiate π_{COM} with the proof of knowledge from [40] (this proof is not a zero-knowledge proof, yet can be modified using standard techniques; [26]).

2.4 Zero-knowledge Proofs

Our protocols employ zero-knowledge proofs of knowledge for the following relations (in the following, $\mathbb{G}$ is a group of prime order):

Proof Type	Protocol	Relation/Language	Reference
ZKPK	π_{DL}	$\mathcal{R}_{\text{DL}} = \{((\mathbb{G}, g, h), x) \mid h = g^x\}$	[44]
ZKPK	π_{DDH}	$\mathcal{R}_{\text{DDH}} = \{((\mathbb{G}, g, g_1, g_2, g_3), x) \mid g_1 = g^x \wedge g_3 = g_2^x\}\}$	[13]
ZK	π_{NZ}	$\text{L}_{\text{NZ}} = \{(\mathbb{G}, g, h, \langle \alpha, \beta \rangle) \mid \exists\, (m \neq 0, r)$ s.t. $\alpha = g^r,\ \beta = h^r g^m\}$	Section 2.4.1

2.4.1 Zero-Knowledge Proof for L_{NZ}

We use standard techniques for constructing a zero-knowledge proof for the language of encryptions $\langle \alpha, \beta \rangle$ of non-zero exponents of g:

$$\text{L}_{\text{NZ}} = \{(\mathbb{G}, g, h, \langle \alpha, \beta \rangle) \mid \exists\, (m \neq 0, r) \text{ s.t. } \alpha = g^r,\ \beta = h^r g^m\}.$$

The construction is based on a zero-knowledge protocol π_{MULT} for the language

$$\text{L}_{\text{MULT}} = \left\{ (\mathbb{G}, g, h, c_1, c_2, c_3) \mid \exists\, m, m' \in \mathbb{Z}_q \text{ s.t. } c_1, c_2, c_3 \text{ are encryptions of } g^m, g^{m'}, g^{mm'} \text{ resp.} \right\}.$$

π_{MULT} is a modification of a protocol by Damgård and M. Jurik [15] designed for the Paillier encryption scheme:

Protocol 1 (zero-knowledge proof for L_{MULT}):

- **Joint statement:** *A public key h for El Gamal encryption, and encryptions $\langle \alpha, \beta \rangle, \langle \alpha', \beta' \rangle, \langle \alpha'', \beta'' \rangle$ (where $h, \alpha, \beta, \alpha', \beta', \alpha'', \beta'' \in \mathbb{G}$).*

- **Auxiliary input for the prover:** *Three message-randomness pairs* $(m, r), (m', r'), (m'', r'')$ *such that* $\langle \alpha, \beta \rangle, \langle \alpha', \beta' \rangle$, $\langle \alpha'', \beta'' \rangle$ *are encryptions of of* $g^m, g^{m'}, g^{m''}$ *with randomness* r, r', r'' *respectively, and* $m'' = mm'$.
- **Auxiliary inputs for both parties:** *A prime* p *such that* $p - 1 = 2q$ *for a prime* q*, the description of a group* $\mathbb{G}$ *of order* q *for which the* DDH *assumption holds, and a generator* g *of* $\mathbb{G}$.
- **The protocol:**
  1. V *chooses at random* $c \in \mathbb{Z}_q$ *and sends a commitment* $z = \mathsf{com}(c; y) = g^c h^y$ *to* P*. The parties engage in the protocol* π_{COM} *where* V *proves knowledge of the committed value* c *and the randomness* y*. If* V *fails to prove knowledge of* c*, then* P *aborts.*
  2. P *chooses at random* $a \in \mathbb{Z}_q$ *sets* $b = am'$ *and sends to* V *encryptions* $\langle \alpha_a, \beta_a \rangle$ *and* $\langle \alpha_b, \beta_b \rangle$ *of* g^a *and* g^b *respectively (i.e.,* g^a *is encrypted as* $\langle \alpha_a, \beta_a \rangle = \langle g^{r_a}, h^{r_a} g^a \rangle$ *where* r_a *is chosen at random from* $\mathbb{Z}_q$*, and similarly for* g^b*).*
  3. V *opens the commitment by sending* y, c *to* P*.* P *verifies that the commitment was opened correctly and aborts if this is not the case.*
  4. *Both parties use the homomorphic properties of the scheme to compute an encryption* $\langle \alpha_d, \beta_d \rangle$ *of* g^d *where* $d = cm + a$*.* P *sends* d *to* V*, and proves in zero knowledge that* $(g, h, \alpha_d, \beta_d / g^d)$ *is a Diffie-Hellman tuple, i.e.,* $\langle \alpha_d, \beta_d \rangle$ *is an encryption of* g^d*.*
  5. *Both parties use the homomorphic properties of the scheme to compute an encryption* $\langle \alpha_e, \beta_e \rangle$ *of* g^e *where* $e = dm' - b - cm''$*. Note that if* $m'' = mm'$ *then* $e = dm' - b - cm'' = (cm + a)m' - am' - cm'' = 0$*.* P *proves in zero knowledge that* $(g, h, \alpha_e, \beta_e / g^d)$ *is a Diffie-Hellman tuple, i.e.,* $\langle \alpha_e, \beta_e \rangle$ *is an encryption of* $g^0 = 1$*.*
  6. V *accepts if it accepts in all zero-knowledge proofs.*

Proposition 2.1 *Assume that* com *is a perfectly-hiding commitment scheme. Then, Protocol* 1 *is a zero-knowledge proof for* L_{MULT} *with perfect completeness and negligible soundness error.*

Proof: Our proof is in the hybrid setting where a trusted party computes π_{DDH} and π_{COM}. It is easy to check that all of V's checks pass when interacting with a honest P that follows the protocol, and hence we have perfect completeness. To prove soundness, let a, b be the numbers encrypted in the first two encryptions sent by a. Since V computes the encryption of d, it must be that $d = cm + a$. The first zero-knowledge proof ensures that V indeed learns d. Similarly, since V computes the encryption of e, it must be that $e = dm' - b - cm'' = (cm + a)m' - b - cm'' = c(mm' - m'') + am' - b$. Now, if $mm' \neq m''$ then $e = 0$ only if $c = (b - am')/(mm' - m'')$. As V accepts only if $e = 0$, we get that it accepts with probability at most $1/q$.

Zero knowledge. Construct a simulator $\mathcal{S}$ for V^* as follows. The input to $\mathcal{S}$ is $(g, h, \langle \alpha, \beta \rangle, \langle \alpha', \beta' \rangle, \langle \alpha'', \beta'' \rangle)$, i.e., the encryption of m, m', m'', and, furthermore, $mm' = m''$. The simulator receives V's commitment z, and the inputs c, y to the functionality $\mathcal{R}_{\text{COM}}$. If the functionality returns 0 then $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ computes an encryption $\langle \alpha_e, \beta_e \rangle$ to g^0, chooses $d \in \mathbb{Z}_q$ at random, and computes an encryption $\langle \alpha_d, \beta_d \rangle$ to g^d. Finally, $\mathcal{S}$ uses the homomorphic operations to compute encryption $\langle \alpha_a, \beta_a \rangle$ of g^a where $a = d - cm$ and an encryption $\langle \alpha_b, \beta_b \rangle$ of g^b where $b = dm' - cm''$. S completes the execution of the protocol using encryptions $\langle \alpha_a, \beta_a \rangle$ and $\langle \alpha_b, \beta_b \rangle$, and returning 1 to V in each of the invocation of the zero-knowledge proofs.

V manages to open the commitment to a value that is different from that given as input to $\mathcal{R}_{\text{COM}}$ with only a negligible probability. If that is not the case, then the views are identical (to see that, note that the a and b are distributed identically in both executions). ■

We continue with a zero-knowledge proof π_{NZ} for language L_{NZ}.

Protocol 2 (zero-knowledge proof for $\mathrm{L_{NZ}}$):

- **Joint statement:** *A public key h for El Gamal encryption, and an encryption $\langle \alpha, \beta \rangle$ ($h, \alpha, \beta \in \mathbb{G}$).*
- **Auxiliary input for the prover:** *A pair (m, r) such that $m \neq 0$ and $\langle \alpha, \beta \rangle$ is an encryption of m with randomness r, i.e., $\alpha = g^r$ and $\beta = h^r g^m$.*
- **Auxiliary inputs for both parties:** *A prime p such that $p - 1 = 2q$ for a prime q, the description of a group $\mathbb{G}$ of order q for which the* DDH *assumption holds, and a generator g of $\mathbb{G}$.*
- **The protocol:**
  1. *The prover P chooses a random value $m' \in \mathbb{Z}_q \setminus \{0\}$ and sets $m'' = mm'$. P computes encryptions $\langle \alpha', \beta' \rangle, \langle \alpha'', \beta'' \rangle$ of $g^{m'}, g^{m''}$ respectively , i.e., $\alpha' = g^{r'}, \beta' = h^{r'} g^{m'}$ and $\alpha'' = g^{r''}, \beta'' = h^{r''} g^{m''}$.*
  2. *P, V engage in π_{MULT}, where P proves that the ciphertexts $\langle \alpha, \beta \rangle, \langle \alpha', \beta' \rangle, \langle \alpha'', \beta'' \rangle$ encrypt messages $g^m, g^{m'}, g^{m''}$ such that $m'' = mm'$.*
  3. *P sends m'' and proves that $\langle \alpha'', \beta'' \rangle$ is an encryption of $g^{m''}$ by proving that $(g, h, \alpha'', \beta''/g^{m''})$ is a Diffie-Hellman tuple using π_{DDH}.*
  4. *V accepts if it accepts in the zero-knowledge proof and the received m'' is non-zero.*

Proposition 2.2 *Assuming hardness of the* DDH *problem Protocol 2 is a computational zero-knowledge proof for $\mathrm{L_{NZ}}$ with perfect completeness and negligible soundness.*

Proof: Our proof is in the hybrid setting where a trusted party computes π_{DDH} and π_{MULT}. Completeness follows trivially as all of V's checks pass when interacting with the honest P that computes $m'', \alpha', \beta', \alpha'', \beta''$ correctly. To prove soundness also follows easily noting that a zero m multiplied by any value results a zero, hence P fails in proving that $\langle \alpha'', \beta'' \rangle$ is an encryption of $g^{m''}$ for $m'' \neq 0$.

Zero knowledge. We construct a simulator $\mathcal{S}$ for V^* as follows: The input to $\mathcal{S}$ the encryption of m, i.e., $(g, h, \langle \alpha, \beta \rangle)$. $\mathcal{S}$ chooses m', m'' at random and sends to V their encryptions $\langle \alpha', \beta' \rangle, \langle \alpha'', \beta'' \rangle$. It then emulates the trusted party for π_{MULT} and returns 1 to the verifier for the hybrid executions of π_{MULT}. Finally, $\mathcal{S}$ sends m'' to V, then emulates the trusted party for π_{DDH} and returns 1 to V.

The view of V is $(g, h, \alpha, \beta, \alpha', \beta', \alpha'', \beta'', m'')$, where $\langle \alpha, \beta \rangle, \langle \alpha', \beta' \rangle, \langle \alpha'', \beta'' \rangle$ are encryptions of messages m, m', m''. In the hybrid execution $m'' = mm'$ whereas in the simulated execution m'' is random. The computationally indistinguishability of these views follows directly from the semantic security of the El Gamal encryption scheme. ■

2.5 Balanced Allocation

We employ a scheme for randomly mapping elements into bins, as suggested in [24]. We use the balanced allocation scheme of [3] where elements are inserted into B bins as follows. Let $h_0, h_1 : \{0,1\}^{p(n)} \to [B]$ be two randomly chosen hash functions mapping elements from $\{0,1\}^{p(n)}$ into bins $1, \ldots, B$. An element $x \in \{0,1\}^{p(n)}$ is inserted into the less occupied bin from $\{h_0(x), h_1(x)\}$, where ties are broken arbitrarily. If m elements are inserted, then except with negligible probability over the choice of the hash functions h_0, h_1, the maximum number of elements allocated to any single bin is at most $M = O(m/B + \log\log B)$. Setting $B = \frac{m}{\log\log m}$ we get that $M = O(\log\log m)$.[1] In the protocol we deviate insignificantly from the description above, and let P_1 choose seeds for two pseudorandom functions, that are used as the hash functions h_0, h_1.

[1] A constant factor improvement is achieved using the *Always Go Left* scheme in [45] where $h_0 : \{0,1\}^{p(n)} \to [1, \ldots, \frac{B}{2}], h_1 : \{0,1\}^{p(n)} \to [\frac{b}{2} + 1, \ldots, B]$. An element x is inserted into the less occupied bin from $\{h_0(x), h_1(x)\}$; in case of a tie x is inserted into $h_0(x)$.

2.6 Oblivious PRF Evaluation

We use a protocol π_{PRF} for that obliviously evaluates a pseudorandom function in the presence of a malicious adversary. Let I_{PRF} be the indexing algorithm for a pseudorandom function ensemble, and let $k \leftarrow_R I_{\text{PRF}}(1^n)$ be a sampled key. The functionality F_{PRF} is defined as

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

The PRF may be instantiated with the Naor-Reingold pseudorandom function [38] with the protocol presented in [23] (and proven in [28]). The function is defined as

$$F_{\text{PRF}}((a_0, \ldots, a_n), x) = g^{a_0 \prod_{i=1}^{n} a_i^{x[i]}},$$

where $\mathbb{G}$ is a group of prime order q, g is a generator of $\mathbb{G}$, $a_i \in \mathbb{Z}_q$ and $x = (x[1], \ldots, x[n]) \in \{0,1\}^n$. The protocol involves executing an oblivious transfer for every bit of the input x. Combining this with the fact that n oblivious transfers runs require $11n + 29$ exponentiations using the protocol in [43] (the analysis in [43] includes the cost for generating a common reference string), one gets a constant-round protocol that securely computes $\mathcal{F}_{\text{PRF}}$ in the presence of malicious players using a constant number of exponentiations for every bit of the input x.

3 Secure Set Intersection

We now consider the functionality of *set intersection*, where each party's input consists of a *set*, and the size of the other party's input set. If the set sizes match, then the functionality outputs the intersection of these input sets to P_1. Otherwise P_1 is given $\perp$. More formally:

Definition 3.1 *Let X and Y be subsets of a predetermined domain (wlog, we assume $X, Y \subset \{0,1\}^{p(n)}$ for some polynomial $p()$ such that $2^{p(n)}$ is super-polynomial in n, and that the set elements can be represented as elements of some finite group), the functionality $\mathcal{F}_\cap$ is:*

$$((X, m_Y), (Y, m_X)) \mapsto \begin{cases} (X \cap Y, \lambda) & \text{if } |X| = m_X, |Y| = m_Y \text{ and } X, Y \subseteq \{0,1\}^{p(n)} \\ (\perp, \lambda) & \text{otherwise} \end{cases}$$

In the rest of this section we present in detail our construction for a protocol realizing $\mathcal{F}_\cap$ in the presence of malicious adversaries. We begin by briefly describing the construction of Freedman et al. [24] for semi-honest parties that serves as our starting point.

Secure Set Intersection with Semi-Honest Parties. The main tool used in the construction of [24] is oblivious polynomial evaluation. The basic protocol works as follows:

1. Party P_1 chooses encryption/decryption keys $(pk, sk) \leftarrow G(1^n)$ for a homomorphic encryption scheme (G, E, D) and sends pk to P_2.

2. P_1 computes the coefficients of a polynomial $Q(\cdot)$ of degree m_X, with roots set to the m_X elements of X, and sends the encrypted coefficients to P_2.

3. For each element $y \in Y$ (in random order), party P_2 chooses a random value r (taken from an appropriate set depending on the encryption scheme), and uses the homomorphic properties of the encryption scheme to compute an encryption of $r \cdot Q(y) + y$. P_2 sends the encrypted values to P_1.

4. Upon receiving these encrypted values, P_1 extracts $X \cap Y$ by decrypting each value and then checking if the result is in X. Note that if $y \in X \cap Y$ then by the construction of the polynomial $Q(\cdot)$ we get that $r \cdot Q(y) + y = r \cdot 0 + y = y$. Otherwise, $r \cdot Q(y) + y$ is a random value that reveals no information about y and (with high probability) is not in X.

Note that the communication complexity of this simple scheme is linear in $m_X + m_Y$, as $m_X + 1$ encrypted values are sent from P_1 to P_2 (these are the encrypted coefficients of $Q(\cdot)$), and m_Y encrypted values are sent from P_2 to P_1 (i.e., $Q(y)$ for every $y \in Y$). However, the work performed by P_2 is high, as each of the m_Y oblivious polynomial evaluations includes performing $O(m_X)$ exponentiations, totaling in $O(m_X \cdot m_Y)$ exponentiations.

To save on computational work, Freedman et al. introduced a balanced allocation scheme into the protocol. Loosely speaking, they used the balanced allocation scheme of [3] mentioned above with $B = \frac{m_X}{\log \log m_X}$ bins, each of size $M = O(m_X/B + \log \log B) = O(\log \log m_X)$. Party P_1 now uses the balanced allocation scheme to hash every $x \in X$ into one of the B bins resulting (with high probability) with each bin's load being at most M. Instead of a single polynomial of degree m_X party P_1 now constructs a degree-M polynomial for each of the B bins, i.e., polynomials $Q_1(\cdot), \ldots, Q_B(\cdot)$ such that the roots of $Q_i(\cdot)$ are the elements put in bin i. As some of the bins contain less than M elements, P_1 pads each polynomial with zero coefficients up to degree M. Upon receiving the encrypted polynomials, party P_2 obliviously evaluates the encryption of $r_0 \cdot Q_{h_0(y)}(y) + y$ and $r_1 \cdot Q_{h_1(y)}(y) + y$ for each of the two bins $h_0(y), h_1(y)$ in which y can be allocated, enabling P_1 to extract $X \cap Y$ as above.

Neglecting small constant factors, the communication complexity is not affected as P_1 now sends $BM = O(m_X)$ encrypted values and P_2 replies with $2m_Y$ encrypted values. There is, however, a dramatic reduction in the work performed by P_2 as each of the oblivious polynomial evaluations amounts now to performing just $O(M)$ exponentiations, and hence P_2 performs $O(m_Y \cdot M) = O(m_Y \cdot \log \log m_X)$ exponentiations overall. For completeness we include a description of the protocol by Freedman et al. for semi-honest parties:

Protocol 3 (set-intersection protocol secure in the presence of semi-honest parties):

- **Inputs:** *The input of P_1 is m_Y and a set $X \subseteq \{0,1\}^{p(n)}$ containing m_X items; the input of P_2 is m_X and a set $Y \subseteq \{0,1\}^{p(n)}$ containing m_Y items.*
- **Auxiliary inputs:** *A security parameter 1^n.*
- **The protocol:**

  1. **Key setup:** *P_1 chooses the secret and public keys (sk, pk) for the underlying homomorphic encryption scheme (e.g., Paillier or El Gamal). She sends pk to P_2.*
  2. **Setting the balanced allocation scheme:** *P_1 computes the parameters B, M for the scheme and chooses the seeds for two (pseudo-)random hash functions $h_0, h_1 : \{0,1\}^{p(n)} \to [B]$. She sends B, M, h_0, h_1 to P_2.*
  3. **Creating polynomials for the set X:** *For every $x \in X$, P_1 maps x into the less occupied bin from $\{h_0(x), h_1(x)\}$ (ties broken arbitrarily). Let $\mathcal{B}_i$ denote the set of elements mapped into bin i and let $Q_i(x) \stackrel{\text{def}}{=} \sum_{j=0}^{M} Q_{i,j} \cdot x^j$ denote a polynomial with the set of roots $\mathcal{B}_i$. P_1 encrypts the coefficients of the polynomials and sends the encrypted coefficients to P_2.*
  4. **Substituting in the polynomials:** *Let $y_1, \ldots, y_{m_Y}$ be a random ordering of the elements of set Y. P_2 does the following for all $\alpha \in \{1, \ldots, m_Y\}$:*
     (a) *He sets $\hat{h}_0 = h_0(y_\alpha)$, $\hat{h}_1 = h_1(y_\alpha)$.*
     (b) *He chooses two random elements in the underlying group of the homomorphic encryption scheme r_0, r_1. He then uses the homomorphic properties of the encryption scheme to compute an encryption of $r_0 \cdot Q_{\hat{h}_0}(y_\alpha) + y_\alpha$ and $r_1 \cdot Q_{\hat{h}_1}(y_\alpha) + y_\alpha$. Both encrypted values are sent to P_1.*

5. **Computing the intersection:** *P_1 decrypts each received value. If the decrypted value is in X then P_1 records as part of her local output.*

Note that, since the parties are semi-honest, P_1 outputs $X \cap Y$ with probability negligibly close to 1: (i) For elements $y_\alpha \in X \cap Y$ we get that $Q_{h_0(y_\alpha)}(y_\alpha) = 0$ or $Q_{h_1(y_\alpha)}(y_\alpha) = 0$, hence one of the corresponding encrypted values is y_α itself, and P_1 would record it in its local output. (ii) For $y_\alpha \notin X \cap Y$ we get that $Q_{h_0(y_\alpha)}(y_\alpha) \neq 0$ and $Q_{h_1(y_\alpha)}(y_\alpha) \neq 0$ and hence corresponding encrypted values are two random values $r_0 + y$ and $r_1 + y$ that fall within X with only a negligible probability.

Efficiency. The protocol runs in a constant number of rounds. The communication costs are of sending the encrypted polynomials (BM values) and the encrypted $r_0 \cdot Q_{\hat{h}_0}(y_\alpha) + y_\alpha$ and $r_1 \cdot Q_{\hat{h}_1}(y_\alpha) + y_\alpha$ ($2m_Y$ values). Using the El Gamal or Paillier encryption schemes, the computation costs are of encrypting the polynomials ($O(BM)$ exponentiations) and of obliviously computing the encryptions of $r_0 \cdot Q_{\hat{h}_0}(y_\alpha) + y_\alpha$ and $r_1 \cdot Q_{\hat{h}_1}(y_\alpha) + y_\alpha$ ($O(Mm_Y)$ exponentiations). Overall, we get that the overall communication costs are of sending $O(m_X + m_Y)$ encryptions, and the computation costs are of performing $O(m_X + m_Y \log\log n)$ modular exponentiations.

3.1 Constructing a Protocol for Malicious Parties

We note a couple of issues that need to be addressed in transforming the above protocol for semi-honest parties to a protocol for malicious parties:

1. It is easy for P_1 to construct the B polynomials such that it would learn about elements that are not in the intersection $X \cap Y$. For instance, if $Q_i(\cdot)$ is identically zero then P_1 learns all elements $\{y \in Y : h_0(y) = i \text{ or } h_1(y) = i\}$. Similarly, if the sum of degrees of $Q_1, \ldots, Q_B$ exceeds m_X then P_1 may learn about more than m_X elements in P_2's input.

   To resolve these problems we introduce a zero-knowledge protocol for verifying that $Q_i \not\equiv 0$ for all $i \in \{1, \ldots, B\}$, and $\sum_{i \in \{1,\ldots,B\}} \mathsf{deg}(Q_i) = m_X$.

2. While party P_2 is supposed to send m_Y pairs of encryptions resulting from substituting a value y (known to P_2) in the (encrypted) polynomials $Q_{h_0(y)}$ and $Q_{h_1(y)}$ it may deviate from his prescribed computation. Thus, P_2's input to the protocol may be ill defined. A solution suggested in [24] solves this problem partially, as it deals with the case where each element y is substituted in a single polynomial. This solution avoids the standard usage of zero-knowledge proofs by P_2 that it indeed followed the protocol. Instead, it enables party P_1 to redo the entire computation supposedly carried out by P_2 on y and verify that its outcome is consistent with the messages received from P_2 (this is where the construction uses a random oracle).

   We remove the dependency on the random oracle and present a solution to the case where y is substituted in two polynomials.

3.2 Checking the Polynomials

Our set-intersection protocol utilizes a zero-knowledge proof of knowledge for the relation[2]

$$\mathcal{R}_{\text{POLY}} = \left\{ \left(\{q_{i,j}\}_{i,j}, m_X, pk \right), \left(\{Q_{i,j}, r_{i,j}\}_{i,j} \right) \middle| \begin{array}{c} \forall i,j \; q_{i,j} = E_{pk}(Q_{i,j}; r_{i,j}) \wedge \\ \sum_i \mathsf{deg}(Q_i(\cdot)) = m_X \; \wedge \forall i, \; Q_i(\cdot) \not\equiv 0 \end{array} \right\}$$

[2]We will use the convention that the degree of a polynomial $Q_i(\cdot)$ can be chosen to be any integer j' such that $Q_{i,j} = 0$ for all $j \geq j'$, hence equality with m_X can always be achieved.

where $i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}$.

For completeness, we give the zero-knowledge proof of knowledge for $\mathcal{R}_{\text{POLY}}$. The main ingredient of the protocol is a sub-protocol where for every polynomial $Q_i(\cdot)$ the prover sends a sequence of encrypted values $Z_{i,j}$ where $Z_{i,j} = 1$ for every $0 \leq j \leq \deg(Q_i(\cdot))$, and 0 otherwise. After the prover proves that the values $Z_{i,j}$ were constructed correctly, calculating the sum of degrees easily translates to summing over $Z_{i,j}$ for $1 \leq i \leq B$ and $0 \leq j \leq M$. This can be done using the homomorphic properties of the El Gamal encryption scheme. More formally,

Protocol 4 (zero-knowledge proof of knowledge π_{POLY} for $\mathcal{R}_{\text{POLY}}$):

Joint statement: $B \cdot (M+1)$ *encryptions* $\{q_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}$, *a public-key* $pk = \langle \mathbb{G}, q, g, h\rangle$ *and an integer* m_X.

- **Auxiliary inputs for the prover:** $B\cdot(M+1)$ *pairs* $\{Q_{i,j}, r_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}$ *where* $q_{i,j}$ *is an encryption of* $Q_{i,j}$ *with randomness* $r_{i,j}$.
- **Convention:** *Both parties check every received ciphertext for validity (i.e, that it is in* $\mathbb{G}$*), and abort if an invalid ciphertext is received. Unless written differently,* $i \in \{1,\ldots,B\}$ *and* $j \in \{0,\ldots,M\}$.
- **Notation:** *We abuse notation and write* $E_{pk}(m)$ *(instead of* $E_{pk}(g^m)$*) for the encryption of* g^m. *Using this notation, the El Gamal encryption scheme is additively homomorphic. We note, however, that the result of decrypting* $E_{pk}(m)$ *is* g^m, *i.e.,* $D_{sk}\left(E_{pk}(m)\right) = g^m$.
- **The protocol:**

  1. *For every* $q_{i,j} = \langle \alpha_{i,j}, \beta_{i,j}\rangle$, P *proves the knowledge of* $\log_g \alpha_{i,j}$ *using* π_{DL}.
  2. P *sets* $Z_{i,j} = 1$ *for* $0 \leq j \leq \deg(Q_i(\cdot))$, *and otherwise* $Z_{i,j} = 0$. *The prover* P *computes* $z_{i,j} = E_{pk}(Z_{i,j})$ *and sends* $\{z_{i,j}\}_{i,j}$ *to the verifier* V. *For all* $i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}$, V *chooses* $u_{i,j} \leftarrow_R \mathbb{Z}_q$ *and sends* $\{u_{i,j}\}_{i,j}$ *to* P. *For* $i \in \{1,\ldots,B\}$, P *performs the following:*
     (a) P *proves that* $Z_{i,0}, Z_{i,1}, \ldots, Z_{i,M}$ *is monotonically non-increasing, i.e., that* $Z_{i,j} = 0$ *and* $Z_{i,j+1} = 1$ *does not happen for any value of* $j \in \{0,\ldots,M-1\}$. *For that,* P *and* V *compute an encryptions of* $Z_{i,j} - Z_{i,j+1}$ *and* $1 - Z_{i,j} - Z_{i,j+1}$, *and* P *proves that* $Z_{i,j} - Z_{i,j+1} \in \{0,1\}$ *by showing that one of these encryptions denotes a Diffie-Hellman tuple.*[3]
     (b) *It completes the proof that the values* $Z_{i,j}$ *were constructed correctly by proving (similarly to Step 2a above) for all* i, j *that one of the encryptions* $\{q_{i,j}, z'_{i,j}\}$ *is an encryption of zero, where* $z'_{i,j}$ *is an encryption of* $1 - Z_{i,j}$.[4]
     (c) *It proves that for all* i, *that* $Q_i(\cdot)$ *is not identically zero (i.e., that* $Q_{i,j} \neq 0$ *for some* j*): Both parties use the homomorphic properties of the encryption scheme to compute an encryption* $v_i = E_{pk}(\sum_{j=0}^{M} u_{i,j} \cdot Q_{i,j})$. P *then proves that* v_i *is not an encryption of zero using* π_{NZ}.
     (d) *Finally, to prove that the sum of degrees of the polynomials* $\{Q_i(\cdot)\}$ *equals* m_X, *both parties compute an encryption* t *of* $T = \sum_{i,j} Z_{i,j} - B - m_X$. *Then* P *proves that* $(pk, E_{pk}(T))$ *is a Diffie-Hellman tuple using* π_{DDH}.
  3. V *verifies all the zero-knowledge proofs and decryptions. If any of the verifications fails,* V *outputs 0, otherwise, it outputs 1.*

Note that Protocol π_{POLY} runs in a constant number of rounds because each of the zero-knowledge proofs can be implemented in constant rounds and can be invoked in parallel for multiple instances. The

[3] This proof is a simple extension of the standard proof for $\mathcal{R}_{\text{DDH}}$ using a general technique. In particular, the prover separates the challenge c it is given by the verifier into two values; c_1 and c_2 such that $c = c_1 \oplus c_2$. Assume w.l.o.g. that it does not have a witness for the first statement, then it always chooses c_1 in which it knows how to complete the proof (similarly to what the simulator for π_{DDH} does), and uses its witness for the other statement to complete the second proof on a giving challenge c_2. Note that the verifier cannot distinguish whether the prover knows the first of the second witness. See [12] for more details.

[4] We ignore the case in which both encryptions are zero encryptions since it means that P may not count a zero.

parties compute and exchange $O(BM) = O(m_X)$ encryptions and execute $O(BM) = O(m_X)$ zero-knowledge proofs (for $\pi_{\text{DDH}}, \pi_{\text{NZ}}$). Overall, these amount to performing $O(m_X)$ exponentiations and exchanging $O(m_X)$ group elements.

We prove security in the following theorem,

Theorem 3.2 *Assume that π_{DL}, π_{DDH} and π_{NZ} are as described above and that (G, E, D) is the El Gamal encryption scheme. Then π_{POLY} (Protocol 4) is a computational zero-knowledge proof of knowledge for $\mathcal{R}_{\text{POLY}}$ with perfect completeness.*

Proof: We first show perfect completeness. Clearly, V always accepts and outputs 1 since the parties sum up the degrees of the polynomials from the set $\{Q_i(\cdot)\}_i$.

Zero knowledge. Let V^* be an arbitrary probabilistic polynomial-time strategy for V. Then a simulator $\mathcal{S}_{\text{POLY}}$ for this proof can be constructed using the simulators $\mathcal{S}_{\text{DL}}$, $\mathcal{S}_{\text{DDH}}$ and $\mathcal{S}_{\text{NZ}}$ from the corresponding proofs of π_{DL}, π_{DDH} and π_{NZ}. That is, $\mathcal{S}_{\text{POLY}}$ invokes V^* and sends it the sets of encryptions $\{z_{i,j}\}_{i,j}$ and $\{m_{i,j}\}_{i,j}$, that actually are encryptions of zeros. It completes the execution such that every time it is needed to play a prover in a zero-knowledge proof, it runs the appropriate simulator.

We show below that the output distributions in the real and the simulated proofs are computationally indistinguishable. The proof uses a sequence of hybrid games.

Game H_0**:** The simulated execution.

Game H_1**:** In this game the simulator $\mathcal{S}_1$ is given as input P's witness $\{Q_{i,j}, r_{i,j}\}_{i,j}$. $\mathcal{S}_1$ works exactly like $\mathcal{S}$, except that instead of sending zero encryptions, it computes the sets $\{z_{i,j}\}_{i,j}$ as in the real proof. Then the output distributions in the simulation and in game Game H_1 is computationally indistinguishable via a standard reduction to the security of (G, E, D).

Games $\text{H}_2 - \text{H}_5$**:** In this series of games we replace every invocation of a simulator with the corresponding real prover. The proof that the output distribution of every two consecutive games is computationally indistinguishable is by reduction to the zero-knowledge property of these proofs.

Note that the last game is identical to the real proof and thus the proof is concluded.

Knowledge extraction. Let $P^*_{x,\zeta,\rho}$ be an arbitrary prover machine where $x = (\{q_{i,j}\}_{i,j}, m_X, pk)$, ζ is an auxiliary input and ρ is $P^*_{x,\zeta,\rho}$'s random tape. In the following analysis, we neglect the probability that each of the following event occurs, as it is negligible: (i) In one of the zero-knowledge proofs V accepts a false statement made by $P^*_{x,\zeta,\rho}$; (ii) For some $i \in \{1, \ldots, B\}$ $\sum_{j=0}^{M} u_{i,j} \cdot Q_{i,j} = 0$ although $\sum_{j=0}^{M} Q_{i,j} \neq 0$ (due to the randomness of $\{u_{i,j}\}_{i,j}$).

Conditioned on the above, it follows immediately that none of the polynomials $Q_i(\cdot)$ is identically zero (otherwise, $P^*_{x,\zeta,\rho}$ convinces V on a false statement for L_{NZ}). It is left to show that $\sum_i \mathsf{deg}(Q_i(\cdot)) - B = m_X$. For all i and j let $Z_{i,j}$ denote the decrypted value of $z_{i,j}$, and note that $Z_{i,j} \in \{0, 1\}$ (if computed honestly) because either $Z_{i,j} = 0$ or $1 - Z_{i,j} = 0$. Noting that the value $a - b \in \{0, 1\}$ excludes the case where $a = 0$ and $b > 0$, we get that that for all i the sequence $Z_{i,0}, \ldots, Z_{i,M}$ is monotonically non-increasing, and finally, as one of these values $\{(1-Z_{i,j}), Q_{i,j}\}$ equals zero, it must be that $Z_{i,j} = 1$ whenever $Q_{i,j} \neq 0$.[5] Combining these two last observations with $Q_i(\cdot) \not\equiv 0$, we get that $\sum_j Z_{i,j} \geq \mathsf{deg}(Q_i(\cdot)) + 1$. Furthermore, we have that $\sum_{i,j} Z_{i,j} - B = m_X$, as proven in Step 2d, we therefore get the required bound on the sum of degrees.

[5] We ignore the case in which $Z_{i,j} > 1$ since it must be that $Q_{i,j} = 0$ and so, P counts zero elements which only increases the overall sum.

It remains to show the existence of a knowledge extractor K which works as follows. It first plays the verifier in π_{POLY} and aborts if it does not accept the proof. If it accepts the proof, then it runs the knowledge extractor for π_{DL} in order to obtain $r_{i,j} = \log_g \alpha_{i,j}$ for all i, j. It then sets $Q_{i,j} = (\beta_{i,j})/(\alpha_{i,j})^{r_{i,j}}$. If K does not succeed in extracting, it outputs fail. (Doing the above naively as described may result in K running in super-polynomial time. This can be solved using the witness-extended emulator described in [34].) ■

3.3 Secure Set-Intersection in the Presence of Malicious Adversaries

We now get to the main contribution of this work – a protocol that securely computes $\mathcal{F}_\cap$ in the presence of malicious adversaries, in the standard model. The main ingredient is a subtle combination of an oblivious pseudorandom function evaluation protocol and a perfectly hiding commitment scheme. Somewhat counter intuitively, the oblivious PRF need not be committed (in a sense that the same key is being reused) – the proof of security shows that although party P_2 that controls the key may change it between invocations, this does not get him any advantage.

The oblivious PRF is used to save on using a (generic) zero-knowledge protocol for party P_2's adherence to the protocol. Recall that in the protocol of Freedman et al. P_1 learns for every $y \in Y$ two values: $r_0 \cdot Q_{h_0(y)}(y) + y$ and $r_1 \cdot Q_{h_1(y)}(y) + y$ where r_0, r_1 are randomly distributed. If $y \notin X$ then both values are random, and reveal no information about y. If, $y \in X$ then one of these values equals y. In our protocol, the 'payload' y of this computation is replaced by a secret s. The result of the polynomial evaluation step is, hence, that if $y \notin X$ then P_1 learns no information about s, and if $y \in X$ then P_1 learns s.

The crux of our construction is that the strings r_0, r_1 (as well as other) are not really random. These are pseudorandom strings that are directly derived from $F_{\text{PRF}}(k, s)$. What we get, is that if $y \notin X$ then P_1 learns nothing about s or y. If, on the other hand, $y \in X$ then P_1 learns s, and furthermore after P_1 invokes the oblivious PRF protocol, she can recover y and check that the computations P_2 performed based on the other 'random' strings were performed correctly.

A complication arises as P_2 (who selects the key k for the PRF) computes $F_{\text{PRF}}(k, s)$ by himself, and hence it is impossible for the simulator to extract s from this computation. We thus provide the simulator with an alternative means of extracting s (and also the corresponding y value) by having P_2 commit to both. To guarantee independence of inputs (i.e., that P_1 would not be able to choose his inputs depending on P_2's commitment or vise versa), this commitment is perfectly hiding and is performed *before* P_1 sends the encrypted polynomials representing her input set X.

We continue with a formal description of the protocol. A high level description is presented in Figure 1.

Protocol 5 ($\pi_\cap$ – secure set-intersection):

- **Inputs:** *The input of P_1 is m_Y and a set $X \subseteq \{0,1\}^{p(n)}$ containing m_X items; the input of P_2 is m_X and a set $Y \subseteq \{0,1\}^{p(n)}$ containing m_Y items (hence, both parties know m_X and m_Y).*
- **Auxiliary inputs:** *A security parameter 1^n, a prime q' such that $q' = 2q + 1$ for a prime q. The group $\mathbb{G}$ is the subgroup of quadratic residues modulo q' and g is a generator of $\mathbb{G}$.*
- **Convention:** *Both parties check every received ciphertext for validity (i.e, that it is in $\mathbb{G}$), and abort otherwise.*
- **The protocol:**

  1. **Key setup for the encryption and commitment schemes:** *P_1 chooses $t, t' \leftarrow_R \mathbb{Z}_q$, sets $h = g^t, h' = g^{t'}$ and sends h, h' to P_2. The key for the Pedersen commitment scheme is h. The public and private keys for the El Gamal scheme are $pk = h'$ and $sk = t'$. P_1 proves knowledge of $\log_g h$ and $\log_g h'$ using the zero-knowledge proof of knowledge for $\mathcal{R}_{\text{DL}}$.*
  2. **Setting the balanced allocation scheme:** *P_1 computes the parameters B, M for the scheme and chooses the seeds for two (pseudo-)random hash functions $h_0, h_1 : \{0,1\}^{p(n)} \to [B]$. She sends B, M, h_0, h_1 to P_2 that checks that the parameters B, M were computed correctly, and aborts otherwise.*

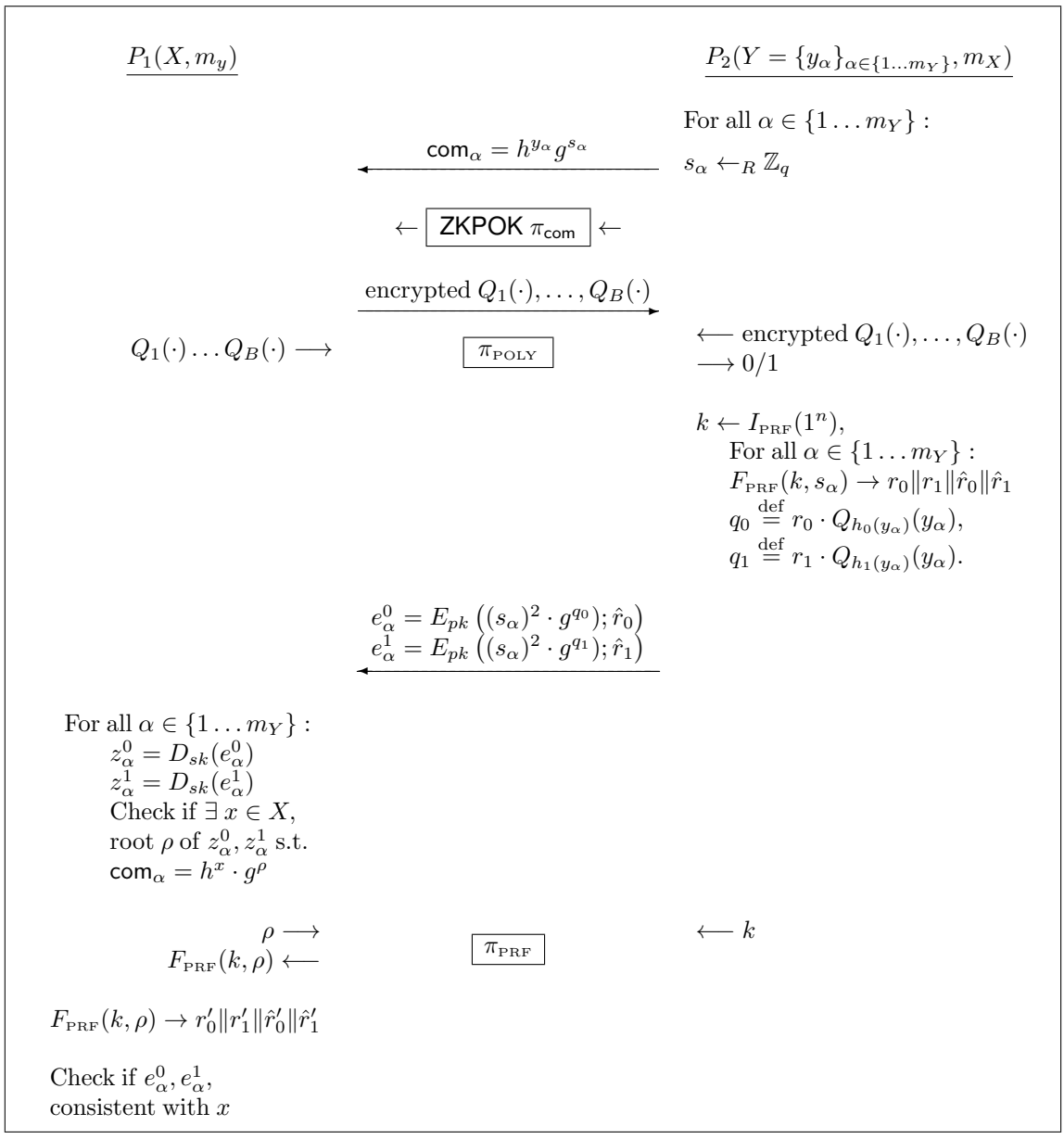

Figure 1: A high-level diagram of $\pi_\cap$.

3. P_2 **commits to his input set:** *Let $y_1, \ldots, y_{m_Y}$ be a random ordering of the elements of set Y. For all $\alpha \in \{1, \ldots, m_Y\}$, P_2 chooses $s_\alpha \leftarrow_R \mathbb{Z}_q$ and sends $\mathsf{com}_\alpha = \mathsf{com}(y_\alpha; s_\alpha) = h^{y_\alpha} g^{s_\alpha}$ to P_1. P_2 then proves the knowledge of y_α and s_α by invoking the zero-knowledge proof of knowledge for $\mathcal{R}_{\text{COM}}$.*
4. P_1 **Creates the polynomials representing her input set:** *For every $x \in X$, P_1 maps x into the less occupied bin from $\{h_0(x), h_1(x)\}$ (ties broken arbitrarily). Let $\mathcal{B}_i$ denote the set of elements mapped into bin i. P_1 constructs a polynomial $Q_i(x) \stackrel{\text{def}}{=} \sum_{j=0}^{M} Q_{i,j} \cdot x^j$ of degree at most M whose set of roots is $\mathcal{B}_i$.[6] P_1 encrypts the polynomials' coefficients, setting $q_{i,j} = E_{pk}(g^{Q_{i,j}}; r_{i,j})$, and sends the encrypted coefficients to P_2.*
5. **Checking the polynomials:** *P_1 and P_2 engage in a zero-knowledge execution π_{POLY} for which P_1 proves that the sets $\{q_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$ were computed correctly, using its witness $\{Q_{i,j}, r_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$. If the outcome is not 1 then P_2 aborts.*
6. **Evaluating the polynomials:** *P_2 chooses $k \leftarrow I_{\text{PRF}}(1^n)$. Then, P_2 performs the following for all $\alpha \in \{1, \ldots, m_Y\}$:*
   (a) *P_2 computes $F_{\text{PRF}}(k, s_\alpha)$ and parses the result to obtain pseudorandom strings r_0, r_1, $\hat{r}_0$, $\hat{r}_1$ of appropriate lengths for their usage below (i.e., $r_0 \| r_1 \| \hat{r}_0 \| \hat{r}_1 = F_{\text{PRF}}(k, s_\alpha)$).*
   (b) *He sets $\hat{h}_0 = h_0(y_\alpha)$ and $\hat{h}_1 = h_1(y_\alpha)$.*
   (c) *He uses the homomorphic properties of the encryption scheme to evaluate $e^0_\alpha = E_{pk}(((s_\alpha)^2 \bmod q') \cdot g^{r_0 \cdot Q_{\hat{h}_0}(y_\alpha)}; \hat{r}_0)$ and $e^1_\alpha = E_{pk}(((s_\alpha)^2 \bmod q') \cdot g^{r_1 \cdot Q_{\hat{h}_1}(y_\alpha)}; \hat{r}_1)$ (where $\hat{r}_0, \hat{r}_1$ denote the randomness used in the re-encryptions). Then he sends e^0_α, e^1_α to P_1.*
7. **Computing the intersection:** *For each $\alpha \in \{1, \ldots, m_Y\}$:*
   (a) *P_1 computes $z^0_\alpha = D_{sk}(e^0_\alpha)$ and $z^1_\alpha = D_{sk}(e^1_\alpha)$. For each of the (up to four) roots ρ of z^0_α, z^1_α (roots are computed modulo $q' = 2q + 1$ and the result is considered only if it falls within $\mathbb{Z}_q$), she checks if $\mathsf{com}_\alpha / g^\rho$ coincides with h^{x_α} for some $x_\alpha \in X$ (this can be done efficiently by creating a hash table for set $\{h^x : x \in X\}$), and if this is the case sets $\hat{s}_\alpha$ to the corresponding root and marks α.*
   (b) *P_1 and P_2 engage in an execution of the protocol for F_{PRF}. If α is marked, then P_1 enters $\hat{s}_\alpha$ as input, and otherwise she enters a zero. P_2 enters k as input. Let $r'_0 \| r'_1 \| \hat{r}'_0 \| \hat{r}'_1$ denote P_1's output from this execution.*
   (c) *If α is marked, then P_1 checks that $e^0_\alpha = E_{pk}((\hat{s}_\alpha)^2 \cdot g^{r'_0 \cdot Q_{h_0(x_\alpha)}(x_\alpha)}; \hat{r}'_0)$, and $e^1_\alpha = E_{pk}((\hat{s}_\alpha)^2 \cdot g^{r'_1 \cdot Q_{h_1(x_\alpha)}(x_\alpha)}; \hat{r}'_1)$ result from applying the homomorphic operations on the encrypted polynomials and randomness r'_0, r'_1. If all checks succeed P_1 records x_α as part of her output.*

A word of explanation is needed for the computation done in Step 6. A natural choice for the payload is s_α itself. However, $s_\alpha \in \mathbb{Z}_q$ whereas the message space of the El Gamal encryption is $\mathbb{G}$. Noting that $\mathbb{Z}_q \subset \mathbb{Z}^*_{q'}$ (neglecting $0 \in \mathbb{Z}_q$), and that by squaring an element of $\mathbb{Z}_{q'}$ we get an element of $\mathbb{G}$, we get that treating s_α as an element of $\mathbb{Z}^*_{q'}$ and computing $(s_\alpha)^2 \bmod q'$ we get an element of $\mathbb{G}$. In this mapping, (up to) two elements of $\mathbb{Z}_q$ share an image in $\mathbb{G}$, and hence in Step 7a we need to recover and check both pre-images.

Before getting into the proof of security, we observe that if both parties are honest, then P_1 outputs $X \cap Y$ with probability negligibly close to one. In this case, if for an element $y_\alpha \in Y$ is holds that $y_\alpha \in X$ then one of $Q_{h_0(y_\alpha)}(y_\alpha)$, $Q_{h_1(y_\alpha)}(y_\alpha)$ is zero, and otherwise none of $Q_{h_0(y_\alpha)}(y_\alpha)$, $Q_{h_1(y_\alpha)}(y_\alpha)$ is zero. We get:

1. If $y_\alpha \in X \cap Y$ then one of e^0_α, e^1_α encrypts $(s_\alpha)^2 \bmod q'$. Hence, there exists a root ρ of z^0_α, z^1_α such that $\mathsf{com}_\alpha / g^\rho$ coincides with h^{x_α} for some $x_\alpha \in X$, resulting in P_1 marking α. Furthermore, as $r_0, r_1, \hat{r}_0, \hat{r}_1$ are derived from $F_{\text{PRF}}(k, s_\alpha)$, the check done by P_1 in Step 7 succeeds and P_1 records y_α in her output.

[6] If $B_i = \emptyset$ then P_1 sets $Q_i(x) = 1$. Otherwise, if $|\mathcal{B}_i| < M$ then P_1 sets the $M + 1 - |\mathcal{B}_i|$ highest-degree coefficients of $Q_i(\cdot)$ to zero.

2. If $y_\alpha \notin X \cap Y$ then none of e_α^0, e_α^1 encrypts $(s_\alpha)^2$ mod q' and (except for a negligible probability) $\mathsf{com}_\alpha / g^\rho$ coincides with h^{x_α} for no root ρ of z_α^0, z_α^1 and $x_\alpha \in X$. Hence, P_1 does not mark α, and y_α is not considered in Step 7, and not included in P_1's output.

Theorem 3.3 *Assume that π_{DL}, π_{PRF} and π_{POLY} are as described above, that (G, E, D) is the El Gamal encryption scheme, and that* com *is a perfectly-hiding commitment scheme. Then $\pi_\cap$ (Protocol 5) securely computes $\mathcal{F}_\cap$ in the presence of malicious adversaries.*

Proof: We separately prove security in the case that P_1 is corrupted and the case that P_2 is corrupted. For the case where no parties are corrupted, we have demonstrated that the output is correct. Our proof is in a hybrid model where a trusted party is used to compute the ideal functionality F_{PRF}, and the zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathrm{POLY}}$, $\mathcal{R}_{\mathrm{COM}}$ and $\mathcal{R}_{\mathrm{DL}}$.

P_1 is corrupted. Let $\mathcal{A}$ denote an adversary controlling P_1. We construct a simulator $\mathcal{S}$ that commits to a set of m_Y arbitrary elements, extracts $\mathcal{A}$'s input $\widetilde{X}$ and then decommits some of these commitments into $\widetilde{X} \cap Y$ (this is where we use the property of the Pedersen commitment that knowing $\log_g h$ it can be opened to any value). A more formal description follows:

1. $\mathcal{S}$ is given X, m_Y and $\mathcal{A}$'s auxiliary input and invokes $\mathcal{A}$ on these inputs. $\mathcal{S}$ sets $m_X = |X|$.
2. $\mathcal{S}$ receives from $\mathcal{A}$, $((\mathbb{G}, g, q, h), t)$ and $((\mathbb{G}, g, q, h'), t')$ for functionality $\mathcal{R}_{\mathrm{DL}}$ and records t, t' only if $h = g^t$ and $h' = g^{t'}$. Otherwise it aborts, sending $\perp$ to the trusted party for $\mathcal{F}_\cap$.
3. $\mathcal{S}$ receives from $\mathcal{A}$ the parameters B, M and the seeds for the two (pseudo-)random hash functions $h_0, h_1 : \{0,1\}^{p(n)} \to [B]$ used in the balanced allocation scheme. If the parameters B, M were not computed correctly, $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_\cap$ and aborts.
4. For all $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ chooses $(\hat{y}_\alpha, \hat{s}_\alpha) \leftarrow_R \mathbb{Z}_q \times \mathbb{Z}_q$ and computes $\mathsf{com}_\alpha = h^{\hat{y}_\alpha} g^{\hat{s}_\alpha}$. $\mathcal{S}$ sends com_α to $\mathcal{A}$ and emulates the ideal computation of $\mathcal{R}_{\mathrm{COM}}$ by sending to $\mathcal{A}$ the value 1.
5. $\mathcal{S}$ receives from $\mathcal{A}$ the encrypted polynomials $\{q_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$.
6. $\mathcal{S}$ receives from $\mathcal{A}$ its input $\{Q_{i,j}, r_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$ for the ideal computation of $\mathcal{R}_{\mathrm{POLY}}$. If the conditions for outputting $(\lambda, 1)$ are not met then $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_\cap$ and aborts.
7. $\mathcal{S}$ sets $\widetilde{X} = \cup_{i=1}^{B} \{x : Q_i(x) = 0 \wedge (h_0(x) = i \vee h_1(x) = i)\}$ and completes $\widetilde{X}$ to size m_X by adding distinct random elements from $\{0,1\}^{p(n)}$. $\mathcal{S}$ sends $\widetilde{X}$ to the trusted party for $\mathcal{F}_\cap$ and receives as answer a set $Z = \widetilde{X} \cap Y$. $\mathcal{S}$ sets $\widetilde{Y}$ to Z and completes $\widetilde{Y}$ to the size m_Y by adding random elements from $\{0,1\}^{p(n)}$ (outside of $\widetilde{X}$).
8. Let $y_1, \ldots, y_{m_Y}$ be a random ordering of the elements of set $\widetilde{Y}$. For $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ considers each commitment com_α in conjunction with y_α and performs the following:
   (a) $\mathcal{S}$ computes a value s_α such that $\mathsf{com}_\alpha = h^{y_\alpha} g^{s_\alpha}$.[7]
   (b) $\mathcal{S}$ chooses a random string r_α (of the outcome length of the pseudorandom function) and records the pair (s_α, r_α).
   (c) $\mathcal{S}$ parses r_α as $r_0 \| r_1 \| \hat{r}_0 \| \hat{r}_1$ and completes Step 6 of the protocol playing the role of the honest P_2.

[7] Recall that $\mathcal{S}$ knows $t = \log_g h$ and thus is able to decommit com_α into any value of its choice. Moreover, s_α is truly random since the equation $h^{y_\alpha} g^{s_\alpha} = h^{\hat{y}_\alpha} g^{\hat{s}_\alpha}$, yields $t \cdot y_\alpha + s_\alpha = t \cdot \hat{y}_\alpha + \hat{s}_\alpha$, or equivalently $s_\alpha = t \cdot (\hat{y}_\alpha - y_\alpha) + \hat{s}_\alpha$.

9. $\mathcal{S}$ emulates for m_Y times the ideal computation of F_{PRF} as follows: One pair (s_α, r_α) is considered for each emulation of F_{PRF}, in the order in which the pairs were recorded. If it happens that $\mathcal{A}$ enters a value s such that $s = s_\alpha$, then $\mathcal{S}$ returns r_α, otherwise, $\mathcal{S}$ returns a randomly chosen string r of the same length.

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

In the following analysis we denote by $\widetilde{X}$ the set of roots as extracted by $\mathcal{S}$ in Step 7 of the simulation (before $\mathcal{S}$ completes it to size m_X). We ignore the event for which $\mathcal{S}$ completes $\widetilde{X}$ with an element $\tilde{x}$ such that $\tilde{x} \in Y$.

Recall that we compare the simulated execution to a hybrid execution where a trusted party is used to compute the ideal functionality F_{PRF}, and the zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathrm{POLY}}$, $\mathcal{R}_{\mathrm{COM}}$ and $\mathcal{R}_{\mathrm{DL}}$. To prove that $\mathcal{A}$'s output in the hybrid and simulated executions are computationally close we construct a sequence of hybrid games and show that the corresponding random variables $\mathrm{H}_\ell^{\mathcal{A}(z)}(X, Y, n)$ that consist of the output of $\mathcal{A}$ in hybrid game H_ℓ are computationally close.

Game H_0**:** The simulated execution.

Game H_1**:** The simulator $\mathcal{S}_1$ does not interact with the trusted party for $\mathcal{F}_\cap$ and is given P_2's real input Y instead. $\mathcal{S}_1$ works exactly like $\mathcal{S}$ except that in Step 8 of the simulation it considers the commitments com_α in conjunction with the elements of Y (whereas $\mathcal{S}$ uses $\widetilde{Y}$). The only difference between these games is that $\mathcal{S}$ uses random values to complete Z to size m_Y, whereas $\mathcal{S}_1$ uses $Y \setminus \widetilde{X}$. In particular, the difference between the two executions is that in Game H_0 members of $\widetilde{Y}' = \widetilde{Y} \setminus (Y \cap \widetilde{X})$ are used where members of $Y' = Y \setminus \widetilde{X} = Y \setminus (Y \cap \widetilde{X})$ are used in Game H_1.

We claim that the distributions in these two executions are computationally close. Let $y \in Y'$ (resp. $y \in \widetilde{Y}'$) be the α element considered in Game H_1 (resp. Game H_0), then $\mathcal{A}$ receives e_α^0 and e_α^1. Note that e_α^0, e_α^1 are encryptions of random values and hence they do not convey any information about s_α (and hence do not affect the probability of guessing s_α). In particular, the guessing of s_α should only be based on retrieving it from com_α, which, *even if y_α is given explicitly*, requires solving the discrete logarithm problem in $\mathbb{G}$.

Game H_2**:** The simulator $\mathcal{S}_2$ is identical to $\mathcal{S}_1$ except that instead of computing every commitment com_α based on a random value $\hat{y}_\alpha$, and then decommitting it into $y_\alpha \in Y$ it commits to elements of Y from the start. As com is a perfectly hiding commitment scheme, $\mathrm{H}_1^{\mathcal{A}(z)}(X, Y, n)$ and $\mathrm{H}_2^{\mathcal{A}(z)}(X, Y, n)$ are identically distributed.

Game H_3**:** The simulator $\mathcal{S}_3$ is given oracle access to a truly random function H_{Func} and hence does not evaluate it by itself. This makes no difference for the output distribution – the random variables $\mathrm{H}_2^{\mathcal{A}(z)}(X, Y, n)$ and $\mathrm{H}_3^{\mathcal{A}(z)}(X, Y, n)$ are identically distributed.

Game H_4**:** The simulator $\mathcal{S}_4$ is given oracle access to a pseudorandom function $F_{\mathrm{PRF}}(k, \cdot)$. The computational indistinguishability of $\mathrm{H}_3^{\mathcal{A}(z)}(X, Y, n)$ and $\mathrm{H}_4^{\mathcal{A}(z)}(X, Y, n)$ follows from the pseudorandomness of F_{PRF}.

Game H_5**:** The hybrid execution. The output distribution in Game H_4 is identical to the output distribution in the hybrid execution since the only difference is that now P_2 computes the PRF evaluations by himself without the help of an oracle.

***P_2* is corrupted.** Let $\mathcal{A}$ denote an adversary controlling P_2 we construct a simulator $\mathcal{S}$ as follows. We construct a simulator $\mathcal{S}$ that sends zero polynomials instead of B polynomials that were computed based on

the real input X. Furthermore, $\mathcal{S}$ does not use the secret key during the execution, but uses the values it extracts from the execution of $\mathcal{R}_{\mathsf{com}}$ to recompute e_α^0, e_α^1. A more formal description follows:

1. $\mathcal{S}$ is given Y, m_X and 1^n and invokes $\mathcal{A}$ on these inputs. $\mathcal{S}$ sets $m_Y = |Y|$.
2. $\mathcal{S}$ chooses $t, t' \leftarrow_R \mathbb{Z}_q$ and sends $\mathcal{A}$ the keys $h = g^t, h' = g^{t'}$. It then emulates the trusted party that computes $\mathcal{R}_{\mathrm{DL}}$ and sends $\mathcal{A}$ the value 1.
3. $\mathcal{S}$ computes the parameters B, M for the balanced allocation scheme and chooses random seeds for the hash functions h_0, h_1. These are then sent to $\mathcal{A}$.
4. $\mathcal{S}$ receives from $\mathcal{A}$ a set of m_Y commitments $\{\mathsf{com}_\alpha\}_{\alpha=1}^{m_Y}$, supposedly of elements in Y. For each commitment com_α that it receives, $\mathcal{S}$ emulates the trusted party that computes $\mathcal{R}_{\mathsf{com}}$. It receives $\mathcal{A}$'s input for the ideal computation of $\mathcal{R}_{\mathsf{com}}$, which is a pair (y_α, s_α). If $(\mathsf{com}_\alpha, (y_\alpha, s_\alpha)) \notin \mathcal{R}_{\mathsf{com}}$ then $\mathcal{S}$ aborts, sending $\perp$ to the ideal functionality for $\mathcal{F}_\cap$.
5. $\mathcal{S}$ sends to $\mathcal{A}$ the encryptions of B zero polynomials.
6. $\mathcal{S}$ emulates the trusted party for $\mathcal{R}_{\mathrm{POLY}}$. It receives from $\mathcal{A}$ a set of BM coefficients and pk. If $\mathcal{A}$'s input is the exact set of encryptions that it received from $\mathcal{S}$ in the previous step, and pk then $\mathcal{S}$ returns 1, otherwise it returns 0.
7. for each $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ receives e_α^0, e_α^1 from $\mathcal{A}$.
8. Recall that for each e_α^0, e_α^1 that it received in the previous simulation step, $\mathcal{S}$ received in Step 4 of the simulation an input (y_α, s_α) to $\mathcal{R}_{\mathsf{com}}$. $\mathcal{S}$ now receives $\mathcal{A}$'s input k for the ideal computation F_{PRF} corresponding to y_α; note that $\mathcal{A}$ may feed $\mathcal{S}$ with a different k in every round of this loop. $\mathcal{S}$ evaluates $r_0 \| r_1 \| \hat{r}_0 \| \hat{r}_1 = F_{\mathrm{PRF}}(k, s_\alpha)$ and checks if e_α^0, e_α^1, are consistent with $r_0, r_1, \hat{r}_0, \hat{r}_1$ when P_2's input is y_α. That is, $\mathcal{S}$ recomputes these encryptions using $s_\alpha, F_{\mathrm{PRF}}(k, s_\alpha)$ and y_α as the honest P_2 would in the real execution, and checks whether the result equals e_α^0, e_α^1. If the check succeeds, $\mathcal{S}$ locally records the value y_α.
9. $\mathcal{S}$ sets $\widetilde{Y}$ to the set of recorded values, and completes $\widetilde{Y}$ to the size m_Y by adding random elements from $\{0,1\}^{p(n)}$.
10. $\mathcal{S}$ sends $\widetilde{Y}$ to the trusted party and outputs whenever $\mathcal{A}$ does.

We note that there are two differences between the simulated and the hybrid executions. First, $\mathcal{S}$ sends the encryptions of B zero polynomials instead of B polynomials that were computed based on the real input X, and second, $\mathcal{S}$ does not use sk during its execution, and instead uses the information $\mathcal{A}$ sends as input to $\mathcal{R}_{\mathsf{com}}$. In the following we define a sequence of hybrid games and denote by the random variable $\mathrm{H}_\ell^{\mathcal{A}(z)}(X, Y, n)$ (for a fixed n) the joint output of $\mathcal{A}$ and P_1 in hybrid game H_ℓ.

Game H_0**:** The simulated execution.

Game H_1**:** The simulator $\mathcal{S}_1$ acts identically to $\mathcal{S}$ except that it does not get to know the secret keys $t = \log_g h$ and $t' = \log_g h'$. The random variables $\mathrm{H}_0^{\mathcal{A}(z)}(X, Y, n)$ and $\mathrm{H}_1^{\mathcal{A}(z)}(X, Y, n)$ are identically distributed as both $\mathcal{S}$ and $\mathcal{S}_1$ do not use t, t'.

Game H_2**:** In this game there is no trusted party and no honest P_1. Instead, the simulator $\mathcal{S}_2$ is given as input P_1's real input X. $\mathcal{S}_2$ works exactly like $\mathcal{S}_1$, except that instead of sending arbitrary polynomials, it computes the polynomials as in the hybrid execution using the set X. In addition, $\mathcal{S}_2$ does not send $\widetilde{Y}$ to the

trusted party, but uses X to compute and output $X \cap \widetilde{Y}$. The proof that $\mathrm{H}_1^{\mathcal{A}(z)}(X,Y,n)$ and $\mathrm{H}_2^{\mathcal{A}(z)}(X,Y,n)$ are computationally indistinguishable is by reduction to the semantic security of the El Gamal encryption scheme.

Game H_3**:** The simulator $\mathcal{S}_3$ acts identically to $\mathcal{S}_2$ except that $\mathcal{S}_3$ is given $t' = \log_g h'$ (but not $t = \log_g h$). The random variables $\mathrm{H}_2^{\mathcal{A}(z)}(X,Y,n)$ and $\mathrm{H}_3^{\mathcal{A}(z)}(X,Y,n)$ are identically distributed as $\mathcal{S}_2$ and $\mathcal{S}_3$ do not use t'.

Game H_4**:** $\mathcal{S}_4$ computes the intersection as in the hybrid execution. To conclude the proof, We show that $\mathrm{H}_3^{\mathcal{A}(z)}(X,Y,n)$ and $\mathrm{H}_4^{\mathcal{A}(z)}(X,Y,n)$ are statistically close. Note that $\mathcal{S}_3, \mathcal{S}_4$ act identically until they get to the computation of the intersection set, where:

- In Game H_3, the simulator $\mathcal{S}_3$ extracts pairs (y_α, s_α) from the zero-knowledge proof of knowledge for $\mathcal{R}_{\mathsf{com}}$ on a commitment com_α. For each such pair it later receives a key k for F_{PRF} and values e_α^0, e_α^1, and adds y_α to $\widetilde{Y}$ if these values are consistent with the randomness $r_0, r_1, \hat{r}_0, \hat{r}_1$ derived from $F_{\mathrm{PRF}}(k, s_\alpha)$. The output is computed as $X \cap \widetilde{Y}$.

- In Game H_4, the simulator $\mathcal{S}_4$ follows the same procedure as in the hybrid execution. That is, $\mathcal{S}_4$ uses sk to decrypt e_α^0 and e_α^1, computes the (up to) four roots ρ and then decides whether to mark α, and if so – it derives $\hat{s}_\alpha, x_\alpha$ and checks that e_α^0, e_α^1 are consistent with $y_\alpha = x_\alpha$ and randomness $r_0, r_1, \hat{r}_0, \hat{r}_1$ derived from $F_{\mathrm{PRF}}(k, \hat{s}_\alpha)$.

Observe first that if an element y_α satisfies the conditions for being included in the input in Game H_3, it also satisfies the conditions for being included in the input in Game H_4: if $\mathcal{S}_3$ outputs an element y_α then it must be that $y_\alpha \in X$, and either e_α^0 or e_α^1 encrypts $(s_\alpha)^2$ mod q', and $\mathsf{com}_\alpha, e_\alpha^0, e_\alpha^1$ were computed according to the protocol specification, and hence $\mathcal{S}_4$ would have outputted it.

Consider now the reverse direction. Let bad denote the event where there exists an element $x \in X$ that $\mathcal{S}_4$ decided to output, but should not have been outputted by $\mathcal{S}_3$. We show that $\Pr[\mathsf{bad}]$ is negligible. Recall that both $\mathcal{S}_3$ and $\mathcal{A}$ do not know $\log_g h$ and that $\mathcal{S}_3$ knows the secret key sk. Note that for bad to occur it must be that for some commitment com_α, (i) $\mathcal{A}$ gives a pair (y_α, s_α) as input to the functionality $\mathcal{R}_{\mathsf{com}}$ (this pair is recorded by $\mathcal{S}_4$), and (ii) one of e_α^0, e_α^1 is decrypted to $(s')^2$ mod q' where $(s')^2$ mod $q' \neq (s_\alpha)^2$ mod q' such that for some $x \in X$ the values e_α^0, e_α^1 are consistent with setting $y = x$ and the randomness obtained from $F_{\mathrm{PRF}}(k, s')$, and furthermore $\mathsf{com}_\alpha = g^x h^{s'}$. We can therefore construct a non-uniform algorithm that given $g, h \leftarrow_R \mathbb{Z}_q$ for prime q, succeeds in computing $y_\alpha, s_\alpha, x, s'$ such that $g^{y_\alpha} h^{s_\alpha} = g^x h^{s'}$, or equivalently succeeds in computing $\log_g h$.

■

3.3.1 Efficiency

We First note that the protocol is constant round (as all its zero-knowledge proofs and subprotocols are constant round). The costs of using current implementations of F_{PRF} on inputs of length $p(n)$ is that of $p(n)$ oblivious transfer invocations [28], and hence of $O(p(n))$ modular exponentiations. We get that the overall communication costs are of sending $O(m_X + m_Y p(n))$ group elements, and the computation costs are of performing $O(m_X + m_Y(\log\log m_X + p(n)))$ modular exponentiations.

3.3.2 Optimizations

Note first that if the functionality is changes to allow P_2 learn the size of the intersection $m_{X\cap Y}$, then, in Step 7b, it is possible to avoid invoking π_{PRF} when α is not marked. This yields a protocol where $O(m_X + m_{X\cap Y} \cdot p(n))$ group elements are sent, and $O(m_X + m_Y \cdot \log\log m_X + m_{X\cap Y} \cdot p(n))$ exponentiation

are computed. When $m_{X \cap Y} \ll m_Y$, this protocol is significantly more efficient than those suggested in [28] for weaker adversarial models. Furthermore, an improved scheme for oblivious pseudorandom function evaluation with overall complexity which is independent of the input length yields a better efficiency as well.

3.4 A Very Efficient Heuristic Construction

Note that we can now modify protocol $\pi_\cap$ to get a protocol in the random oracle model $\pi_\cap^{\mathrm{RO}}$ by performing the following two changes: (1) the computation of $F_{\mathrm{PRF}}(k, s_\alpha)$ performed by P_2 in Step 7b of $\pi_\cap$ is replaced with an invocation of the random oracle, i.e., P_2 computes $\mathcal{H}(s_\alpha)$; and (ii) the execution of the secure protocol for evaluating F_{PRF} by P_1 and P_2 in Step 7 of $\pi_\cap$ is replaced with an invocation of the random oracle by P_1, i.e., no communication is needed, and instead of providing s' to the protocol for F_{PRF}, P_1 computes $\mathcal{H}(s')$.

A typical proof of security in the random oracle model relies on the simulator's ability to record the inputs on which the random oracle is invoked, and the recorded information is used by the simulator for malicious P_2 while recovering his input. In other words, the proof of security relies on the property of the random oracle that the only way to learn any information about $\mathcal{H}(s)$ is to apply $\mathcal{H}$ on a *well defined input* s. Should $\pi_\cap^{\mathrm{RO}}$ be implemented such that the invocations of the random oracle are replaced by a concrete computation of some function, it seems that this proof of security would collapse, even if very strong hardness assumptions are made with respect to this implementation.

Nevertheless, the situation in protocol $\pi_\cap$ is very different. Note, in particular, that the simulator for malicious P_2 cannot monitor P_2's input to F_{PRF} (nor is this notion of inputs to the function well defined). Instead, the simulator extracts s from the zero-knowledge proof of knowledge for the commitment on P_2's inputs in Step 3 of $\pi_\cap$. This is inherited by the modified protocol $\pi_\cap^{\mathrm{RO}}$. Hence, should the random oracle calls in $\pi_\cap^{\mathrm{RO}}$ be replaced with some primitive Gen, the proof of security may still hold with small modifications, given the hardness assumption on Gen (intuitively, some functions of the outcome of $\mathsf{Gen}(s)$ and s should look random). $\pi_\cap^{\mathrm{RO}}$ can hence be viewed as an intermediate step between the protocol in [24] that utilizes a random oracle to cope with malicious parties, and the protocol suggested in the current paper. If the primitive Gen is realized efficiently (e.g., if its computation incurs a constant number of exponentiations), we get an extremely efficient protocol for $\mathcal{F}_\cap$, where the communication costs are of sending $O(m_X + m_Y)$ group elements, and the number of exponentiations is $O(m_X + m_Y \log\log m_X)$.

For the sake of completeness we include a formal description of protocol $\pi_\cap^{\mathsf{Gen}}$, that is identical to protocol $\pi_\cap$ except for the replacing every invocation of $F_{\mathrm{PRF}}(k, \cdot)$ by a computation of $\mathsf{Gen}(\cdot)$. Note that unlike in an invocation of $F_{\mathrm{PRF}}(k, \cdot)$, no communication is needed for computing $\mathsf{Gen}(\cdot)$.

Protocol 6 ($\pi_\cap^{\mathsf{Gen}}$ – secure set-intersection with a "generator"):

- **Inputs:** *The input of P_1 is m_Y and a set $X \subseteq \{0,1\}^{p(n)}$ containing m_X items; the input of P_2 is m_X and a set $Y \subseteq \{0,1\}^{p(n)}$ containing m_Y items (hence, both parties know m_X and m_Y).*
- **Auxiliary inputs:** *A security parameter 1^n, a prime q' such that $q' = 2q + 1$ for a prime q. The group $\mathbb{G}$ is the subgroup of quadratic residues modulo q' and g is a generator of $\mathbb{G}$.*
- **Convention:** *Both parties check every received ciphertext for validity (i.e, that it is in $\mathbb{G}$), and abort otherwise.*
- **The protocol:**

  1. **Key setup for the encryption and commitment schemes:** *P_1 chooses $t, t' \leftarrow_R \mathbb{Z}_q$, sets $h = g^t, h' = g^{t'}$ and sends h, h' to P_2. The key for the Pedersen commitment scheme is h. The public and private keys for the El Gamal scheme are $pk = h'$ and $sk = t'$. P_1 proves knowledge of $\log_g h$ and $\log_g h'$ using the zero-knowledge proof of knowledge for $\mathcal{R}_{\mathrm{DL}}$.*

2. **Setting the balanced allocation scheme:** *P_1 computes the parameters B, M for the scheme and chooses the seeds for two (pseudo-)random hash functions $h_0, h_1 : \{0,1\}^{p(n)} \to [B]$. She sends B, M, h_0, h_1 to P_2 that checks that the parameters B, M were computed correctly, and aborts otherwise.*
3. P_2 **commits to his input set:** *Let $y_1, \ldots, y_{m_Y}$ be a random ordering of the elements of set Y. For all $\alpha \in \{1, \ldots, m_Y\}$, P_2 chooses $s_\alpha \leftarrow_R \mathbb{Z}_q$ and sends $\mathsf{com}_\alpha = \mathsf{com}(y_\alpha; s_\alpha) = h^{y_\alpha} g^{s_\alpha}$ to P_1. P_2 then proves the knowledge of y_α and s_α by invoking the zero-knowledge proof of knowledge for $\mathcal{R}_{\text{COM}}$.*
4. P_1 **Creates the polynomials representing her input set:** *For every $x \in X$, P_1 maps x into the less occupied bin from $\{h_0(x), h_1(x)\}$ (ties broken arbitrarily). Let $\mathcal{B}_i$ denote the set of elements mapped into bin i. P_1 constructs a polynomial $Q_i(x) \stackrel{\text{def}}{=} \sum_{j=0}^{M} Q_{i,j} \cdot x^j$ of degree at most M whose set of roots is $\mathcal{B}_i$.[8] P_1 encrypts the polynomials' coefficients, setting $q_{i,j} = E_{pk}(g^{Q_{i,j}}; r_{i,j})$, and sends the encrypted coefficients to P_2.*
5. **Checking the polynomials:** *P_1 and P_2 engage in a zero-knowledge execution π_{POLY} for which P_1 proves that the sets $\{q_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$ were computed correctly, using its witness $\{Q_{i,j}, r_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$. If the outcome is not 1 then P_2 aborts.*
6. **Evaluating the polynomials:** *P_2 performs the following for all $\alpha \in \{1, \ldots, m_Y\}$:*
   (a) *He sets $\hat{h}_0 = h_0(y_\alpha)$ and $\hat{h}_1 = h_1(y_\alpha)$.*
   (b) *He parses $\mathsf{Gen}(s_\alpha)$ to obtain pseudorandom strings $r_1, r_2, \hat{r}_0, \hat{r}_1$ of appropriate lengths for their usage below (i.e., $r_1 \| r_2 \| \hat{r}_0 \| \hat{r}_1 = \mathsf{Gen}(s_\alpha)$).*
   He uses the homomorphic properties of the encryption scheme to evaluate $e^0_\alpha = E_{pk}(((s_\alpha)^2 \bmod q') \cdot g^{r_0 \cdot Q_{\hat{h}_0}(y_\alpha)}; \hat{r}_0)$ and $e^1_\alpha = E_{pk}(((s_\alpha)^2 \bmod q') \cdot g^{r_1 \cdot Q_{\hat{h}_1}(y_\alpha)}; \hat{r}_1)$ (where $\hat{r}_0, \hat{r}_1$ denote the randomness used in the re-encryptions). Then he sends e^0_α, e^1_α to P_1.
7. **Computing the intersection:** *For each $\alpha \in \{1, \ldots, m_Y\}$:*
   (a) *P_1 computes $z^0_\alpha = D_{sk}(e^0_\alpha)$ and $z^1_\alpha = D_{sk}(e^1_\alpha)$. For each of the (up to four) roots ρ of z^0_α, z^1_α (roots are computed modulo $q' = 2q + 1$ and the result is considered only if it falls within $\mathbb{Z}_q$), she checks if $\mathsf{com}_\alpha / g^\rho$ coincides with h^{x_α} for some $x_\alpha \in X$ (this can be done efficiently by creating a hash table for set $\{h^x : x \in X\}$), and if this is the case sets $\hat{s}_\alpha$ to the corresponding root and marks α.*
   (b) *If α is marked, then P_1 parses $\mathsf{Gen}(\hat{s}_\alpha)$ to obtain $r'_0, r'_1, \hat{r}'_0, \hat{r}'_1$.*
   (c) *P_1 checks that $e^0_\alpha = E_{pk}((\hat{s}_\alpha)^2 \cdot g^{r'_0 \cdot Q_{h_0(x_\alpha)}(x_\alpha)}; \hat{r}'_0)$, and $e^1_\alpha = E_{pk}((\hat{s}_\alpha)^2 \cdot g^{r'_1 \cdot Q_{h_1(x_\alpha)}(x_\alpha)}; \hat{r}'_1)$ result from applying the homomorphic operations on the encrypted polynomials and randomness r'_0, r'_1. If all checks succeed P_1 records x_α as part of her output.*

4 Secure Set Union

In the following section we present a protocol for the set-union problem, where both parties learn the outcome.[9] The functionality we realize is $\mathcal{F}_\cup$, where each party enters a private input set of *distinct* values from a predetermined domain, and the size of the other party's input set. If the set sizes match, then the functionality outputs the union of these input sets to both parties. Otherwise it outputs $\perp$. More formally, let X and Y be subsets of a predetermined domain, then functionality $\mathcal{F}_\cup$ is presented in Figure 2.[10]

Due to technicality arises in the proof, we assume that every element within the domain set $\{0,1\}^{p(n)}$ is represented by $p(n) + 1$ bits, where the additional bit is set to zero. Furthermore, if the parties receive in their output an element where this bit is set to one, they delete it. Note also that from $X \cup Y$ player P_1 can

[8] If $B_i = \emptyset$ then P_1 sets $Q_i(x) = 1$. Otherwise, if $|\mathcal{B}_i| < M$ then P_1 sets the $M + 1 - |\mathcal{B}_i|$ highest-degree coefficients of $Q_i(\cdot)$ to zero.

[9] The trivial protocol where P_1 sends her input to P_2 is secure in the malicious setting for the case where *one* party learns the outcome. The simulator for the corrupted P_2 can send to the trusted party computing the union an empty set (or a random set if the domain is of super polynomial size). With overwhelming probability the trusted party's output enables the simulator to learn the input set of P_1. This is a case where a protocol that is secure for the malicious model is not secure in the semi-honest model.

[10] The definition is in a different format from that of $\mathcal{F}_\cap$ because both parties receive outputs.

Functionality $\mathcal{F}_\cup$

Functionality $\mathcal{F}_\cup$ proceeds as follows, running parties P_1, P_2 and an adversary $\mathcal{S}$.

- Upon receiving a message (X, m_Y) from P_1 and a message (Y, m_X) from P_2, functionality $\mathcal{F}_\cup$ checks first that $|X| = m_X$, $|Y| = m_Y$ and $X, Y \subseteq \{0,1\}^{p(n)}$, and sends $X \cup Y$ to $\mathcal{S}$. Otherwise it halts.
- Upon receiving from $\mathcal{S}$ a message output, functionality $\mathcal{F}_\cup$ sends $X \cup Y$ to P_1 and P_2.

Figure 2: The set union functionality

deduce $Y \setminus X$ and thus $|X \cap Y|$, but not $X \cap Y$. This means that unlike in $\pi_\cap$, where P_1 needs to identify the elements $y_\alpha \in Y$ such that $Q_{\hat{h}_0}(y_\alpha) = 0$ or $Q_{\hat{h}_1}(y_\alpha) = 0$, in $\pi_\cup$ P_1 needs to identify the elements for which $Q_{\hat{h}_0}(y_\alpha) \neq 0$ and $Q_{\hat{h}_1}(y_\alpha) \neq 0$. This is achieved by a sub-protocol enabling P_2 to compute an encryption of $((s_\alpha)^2 \bmod q') \cdot g^{r_z \cdot Z}$ where Z equals the number of zeros in $\{Q_{\hat{h}_0}(y_\alpha), Q_{\hat{h}_1}(y_\alpha)\}$. P_1 can recover s_α and continue as in protocol $\pi_\cup$ if and only if $Z = 0$ (see steps 6 and 7 of protocol $\pi_\cup$).

A second change with respect to $\pi_\cap$ is that P_1 needs to send her local output Y' – the union set – to P_2, and (as the parties are distrusting) prove correctness of the set she sends. There are two parts to this proof, one preventing P_1 from excluding elements from the union set, the other preventing P_1 from including elements that are not in the union set.

1. To prevent excluding elements, P_2 verifies that $Y \subseteq Y'$ and that Y' contains m_X roots relative of the polynomials $\{Q_i(\cdot)\}_i$ (i.e., these roots correspond to the input set X of P_1). This ensures that P_1 has not excluded elements from its computed output.

2. To prevent including elements not represented by the polynomials $\{Q_i(\cdot)\}$, we have the parties compute the size of the set-intersection (information that is anyway revealed from the output), which is then compared with the number of elements in Y that zero these polynomials (see step 9 of protocol $\pi_\cup$). Note that the check by itself does not prevent P_1 from excluding an element $x' \notin X \cap Y$ (where X is defined by the above set of roots) and replacing it with another element x (that is not in the intersection as well).

The construction of our set-union protocol $\pi_\cup$ is similar to that of $\pi_\cap$, and the description below only details the steps that are changed.

4.1 Zero-Knowledge Proof of Knowledge for $\mathcal{R}_{\text{COUNT}}$

$\pi_\cup$ applies a modified version of π_{POLY} (from Section 3.2). In this section we present a zero-knowledge proof of knowledge for this modified relation, defined by

$$\mathcal{R}_{\text{COUNT}} = \left\{ \Big(\{q_i\}_i, t, pk \Big), \Big(\{Q_i, r_i\}_i \Big) \;\middle|\; \begin{array}{c} \forall i \; q_i = E_{pk}(Q_i; r_i) \wedge \\ |\{i : Q_i \neq 0\}| = t \end{array} \right\}$$

This proof counts the non-zero encryptions within $\{q_i\}_i$ and compares the result with t.

Protocol 7 (zero-knowledge proof of knowledge π_{COUNT} for $\mathcal{R}_{\text{COUNT}}$):

Joint statement: *A set of ℓ encryptions $\{q_i\}_i$, a public-key pk and an integer t.*

- **Auxiliary inputs for the prover:** *A set of ℓ values $\{Q_i, r_i\}_i$.*

- **Auxiliary inputs for both:** *A prime p such that $p-1=2q$ for a prime q, the description of a group $\mathbb{G}$ of order q for which the* DDH *assumption holds, and a generator g of $\mathbb{G}$.*
- **Convention:** *Both parties check every received ciphertext for validity (i.e, that it is in $\mathbb{G}$), and abort if an invalid ciphertext is received. Moreover, the parties concatenate the bit zero to every element in their input set. If their output includes an element with this bit set to one, the parties delete this element.*
- **Notation:** *We abuse notation and write $E_{pk}(m)$ (instead of $E_{pk}(g^m)$) for the encryption of g^m. Using this notation, the El Gamal encryption scheme is additively homomorphic. We note, however, that the result of decrypting $E_{pk}(m)$ is g^m, i.e., $D_{sk}\left(E_{pk}(m)\right)=g^m$.*
- **The protocol:**

  1. *For every $q_i=\langle \alpha_i, \beta_i \rangle$, the prover P proves the knowledge of $\log_g \alpha_i$ using π_{DL}.*
  2. *If $Q_i=0$ then P sets $Z_i=1$ and otherwise $Z_i=0$. P computes $z_i=E_{pk}(Z_i)$ and sends $\{z_i\}_i$ to the verifier V.*
  3. *For all i, P performs the following:*
     (a) *For each z_i encrypting a value Z_i it proves that $Z_i \in \{0,1\}$ as follows. Note that both parties can compute an encryption z_i' of $1-Z_i$.[11] P then proves that one of the encryptions $\{z_i, z_i'\}$ is a zero encryption, by proving that either (pk, z_i) or (pk, z_i') is a Diffie-Hellman tuple.*[12]
     (b) *It then uses π_{NZ} to prove that $Q_i+(1-Z_i)\neq 0$ (this excludes the case in which $Q_i=0$ but $Z_i=1$).*
     (c) *P concludes the proof by proving (similarly to Step 3a above) that one of the encryptions $\{q_i, z_i'\}$ is a zero encryption (which excludes the case in which $Q_i\neq 0$ but $Z_i=0$).*
  4. *Finally, to prove that $|\{i: Q_i(\cdot)\neq 0\}|=t$, both parties compute an encryption e of $T=\sum_i Z_i$, and P proves that $(pk, e/E_{pk}(t))$ is a Diffie-Hellman tuple.*
  5. *V verifies all the zero-knowledge proofs. If any of the verifications fails, V outputs 0, otherwise, it outputs 1.*

Theorem 4.1 *ssume that π_{DL}, π_{DDH} and π_{NZ} are as described above and that (G, E, D) is the El Gamal encryption scheme. Then π_{COUNT} (Protocol 7) is a computational zero-knowledge proof of knowledge for $\mathcal{R}_{\mathrm{COUNT}}$ with perfect completeness.*

Due to the similarity to the proof of Proposition 3.2 we omit the details. Note that Protocol π_{COUNT} runs in a constant number of rounds because each of the zero-knowledge proofs can be implemented in constant rounds and can be invoked in parallel, as discussed in Section 3.2. Furthermore, the parties compute and exchange $O(\ell)$ encryptions and execute $O(\ell)$ zero-knowledge proofs (for π_{DL}, π_{DDH}, π_{NZ}).

4.2 Secure Set-Union in the Presence of Malicious Adversaries

We continue with a formal description of the protocol.

Protocol 8 ($\pi_{\cup}$ – secure set union):

- **Inputs:** *The input of P_1 is m_Y and a set $X\subseteq\{0,1\}^{p(n)}$ containing m_X items; the input of P_2 is m_X and a set $Y\subseteq\{0,1\}^{p(n)}$ containing m_Y items (hence, both parties know m_X and m_Y).*
- **Auxiliary inputs:** *A security parameter 1^n, a prime p such that $p-1=2q$ for a prime q, the description of a group $\mathbb{G}$ of order q for which the* DDH *assumption holds, and a generator g of $\mathbb{G}$.*
- **Convention:** *Both parties check every received ciphertext for validity (i.e, that it is in $\mathbb{G}$), and abort if an invalid ciphertext is received.*

[11] Here and below, we assume that P, V agree on an encryption of 1, for which both know the randomness.

[12] See footnote 3 for a discussion regarding this zero-knowledge proof.

- **The protocol:**

  1. **Key setup for the encryption and commitment schemes:** *as in Step 1 of $\pi_\cap$ (protocol 5).*
  2. **Setting the balanced allocation scheme:** *as in Step 2 of $\pi_\cap$ (protocol 5).*
  3. P_2 **commits to his input set:** *as in Step 3 of $\pi_\cap$ (protocol 3).*
  4. P_1 **Creates the polynomials representing her input set:** *as in Step 4 of $\pi_\cap$ (protocol 5).*
  5. **Checking the polynomials:** *as in Step 5 of $\pi_\cap$ (protocol 5).*
  6. **Evaluating the polynomials:** *P_2 chooses $k \leftarrow I_{\mathrm{PRF}}(1^n)$. Then, P_2 performs the following for all $\alpha \in \{1,\ldots,m_Y\}$:*
     (a) *P_2 computes $F_{\mathrm{PRF}}(k, s_\alpha)$ and parses the result to obtain pseudorandom strings $r_0, r_1, \hat{r}_0, \hat{r}_1, r_\oplus, r_z$ of appropriate lengths for their usage below (i.e., $r_0\|r_1\|\hat{r}_0\|\hat{r}_1\|r_\oplus\|r_z = F_{\mathrm{PRF}}(k, s_\alpha)$).*
     (b) *He sets $\hat{h}_0 = h_0(y_\alpha)$ and $\hat{h}_1 = h_1(y_\alpha)$.*
     (c) *He uses the homomorphic properties of the encryption scheme to evaluate $e^0_\alpha = E_{pk}(g^{r_0 \cdot Q_{\hat{h}_0}(y_\alpha)}; \hat{r}_0)$ and $e^1_\alpha = E_{pk}(g^{r_1 \cdot Q_{\hat{h}_1}(y_\alpha)}; \hat{r}_1)$ (where $\hat{r}_0, \hat{r}_1$ denote the randomness used in the re-encryptions). Then he sends e^0_α, e^1_α and $t_\alpha = y_\alpha \oplus r_\oplus$ to P_1.*
  7. **Computing the union:** *P_1 does the following for each received e^0_α, e^1_α:*
     (a) **Oblivious selection:**
        i. *For each $i \in \{0,1\}$, P_1 sets $Z_i = 1$ if $D_{sk}(e^i_\alpha) = g^0 = 1$, and otherwise $Z_i = 0$. It then computes $z_i = E_{pk}(g^{Z_i})$ and sends z_0, z_1 to P_2.*
        ii. *The parties run a proof that the values z_0, z_1 were computed correctly. (They execute Step 3 of the zero knowledge proof π_{COUNT} for which P_1 proves the validity of its computations.)*
        iii. *P_2 computes $e_z = E_{pk}(((s_\alpha)^2 \bmod q') \cdot g^{r_z \cdot (Z_0 + Z_1)})$ using the homomorphic properties of the El Gamal scheme.*
     (b) *If $D_{sk}(e^0_\alpha) \neq g^0$ and $D_{sk}(e^1_\alpha) \neq g^0$, then P_1 sets $s'_\alpha = \rho$ where ρ is a root of e_z that falls within $\mathbb{Z}_q$. Otherwise, s'_α is set to zero.*
     (c) *Then the parties engage in an execution of π_{PRF} for which P_1 enters s'_α and P_2 enters k. Let $r'_0\|r'_1\|\hat{r}'_0\|\hat{r}'_1\|r'_\oplus\|r'_z$ denote P_1's output from this execution. If for $y' = t_\alpha \oplus r'_\oplus$ it holds that $e^0_\alpha, e^1_\alpha, e_z$ are consistent with P_1's output from the* PRF *evaluation, and $\mathsf{com}_\alpha = h^{s'_\alpha} g^{y'}$ then P_1 records y' as part of her output (that is originally set to X).*
  8. **Sending the output to P_2:** *P_1 sends to P_2 the elements in the recorded set (in random order).*
  9. **Verifying the output by P_2:**
     (a) *Let Y' denote the set that P_2 receives from P_1. P_2 verifies first that $Y \subseteq Y'$.*
     (b) *For all $y' \in Y'$, the parties compute the encryptions of $Q_{h_0(y')}(y')$ and $Q_{h_1(y')}(y')$. Next the parties run π_{COUNT} on these values and P_1 proves that the number of zero encryptions within this set equals m_X. To exclude a case for which $Q_{h_0(y')}(y') = Q_{h_1(y')}(y') = 0$, P_1 proves that either $Q_{h_0(y')}(y') \neq 0$ or $Q_{h_1(y')}(y') \neq 0$ using the zero-knowledge proof π_{NZ}.*
     (c) *P_2 learns the size of the intersection with respect to his evaluations from Step 6. That is, the parties set $\xi = |X \cap Y| = m_X + m_Y - |Y'|$. Next, the parties run π_{COUNT} on the evaluations from Step 6 and P_1 proves that the number of zero encryptions within this set equals ξ. The parties run π_{NZ} as in the previous step.*
     (d) *If all the verifications are successfully completed, P_2 locally outputs Y'.*

Note that if both parties are honest, then they output $X \cup Y$ with probability negligibly close to one. Note that in this case, if for an element $y_\alpha \in Y \setminus X$ then $Q_{h_0(y_\alpha)}(y_\alpha) \neq 0$ and $Q_{h_1(y_\alpha)}(y_\alpha) \neq 0$, and otherwise $Q_{h_0(y_\alpha)}(y_\alpha) = 0$ or $Q_{h_1(y_\alpha)}(y_\alpha) = 0$. We get:

1. If $y_\alpha \in Y \setminus X$ then both of e_α^0, e_α^1 encrypt a random element in $\mathbb{G}$. Thus $Z_0 + Z_1 = 0$ and P_1 is able to recover $(s_\alpha)^2 = D_{sk}(e_z) = D_{sk}(E_{pk}((s_\alpha)^2 \bmod q') \cdot g^{r_z \cdot (Z_0 + Z_1)})$. Furthermore, as $r_0, r_1, \hat{r}_0, \hat{r}_1, r_\oplus, r_z$ are derived from $F_{\mathrm{PRF}}(k, s_\alpha)$, the check done by P_1 in Step 7 succeeds and P_1 records y_α in her output.

2. If $y_\alpha \in X \cap Y$ then one of e_α^0, e_α^1 encrypts 0 and thus P_1 cannot recover s_α since $D_{sk}(e_z)$ is a random element in $\mathbb{G}$.

Furthermore, as P_1 computes $Y' = X \cup Y$ correctly, P_2 outputs this set as well. In particular, Y', initially set to X, includes the set $Y \backslash Y \cap X$ as explained above.

Theorem 4.2 *Assume that π_{DL}, π_{NZ}, π_{PRF}, π_{POLY} and π_{COUNT} are as described above, that (G, E, D) is the El Gamal encryption scheme, and that* com *is a perfectly-hiding commitment scheme. Then $\pi_\cup$ securely computes $\mathcal{F}_\cup$ in the presence of malicious adversaries.*

Proof: We separately prove security in the case that P_1 is corrupted and the case that P_2 is corrupted. For the case where no parties are corrupted, we have demonstrated that the output is correct. Our proof is in a hybrid model where a trusted party is used to compute the ideal functionality F_{PRF}, and the zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathrm{COUNT}}$, $\mathcal{R}_{\mathrm{POLY}}$, $\mathcal{R}_{\mathrm{COM}}$ and $\mathcal{R}_{\mathrm{DL}}$.

P_1 is corrupted. Let $\mathcal{A}$ denote an adversary controlling P_1. We construct a simulator $\mathcal{S}$ that commits to a set of m_Y arbitrary elements, extracts $\mathcal{A}$'s input $\widetilde{X}$ and then decommits some of these commitments into $\widetilde{X} \cup Y \backslash \widetilde{X}$ (this is where we use the property of the Pedersen commitment that knowing $\log_g h$ it can be opened to any value). Finally, $\mathcal{S}$ verifies $\mathcal{A}$'s output as in the hybrid execution. A more formal description follows:

1. $\mathcal{S}$ is given X, m_Y and $\mathcal{A}$'s auxiliary input and invokes $\mathcal{A}$ on these inputs. $\mathcal{S}$ sets $m_X = |X|$.

2. $\mathcal{S}$ receives from $\mathcal{A}$, $((\mathbb{G}, g, q, h), t)$ and $((\mathbb{G}, g, q, h'), t')$ for functionality $\mathcal{R}_{\mathrm{DL}}$ and records t, t' only if $h = g^t$ and $h' = g^{t'}$. Otherwise it aborts, sending $\perp$ to the trusted party for $\mathcal{F}_\cap$.

3. $\mathcal{S}$ receives from $\mathcal{A}$ the parameters B, M and the seeds for the two (pseudo-)random hash functions $h_0, h_1 : \{0,1\}^{p(n)} \to [B]$ used in the balanced allocation scheme. If the parameters B, M were not computed correctly, $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_\cap$ and aborts.

4. For all $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ chooses $(\hat{y}_\alpha, \hat{s}_\alpha) \leftarrow_R \mathbb{Z}_q \times \mathbb{Z}_q$ and computes $\mathsf{com}_\alpha = h^{\hat{y}_\alpha} g^{\hat{s}_\alpha}$. $\mathcal{S}$ sends com_α to $\mathcal{A}$ and emulates the ideal computation of $\mathcal{R}_{\mathrm{COM}}$ by sending to $\mathcal{A}$ the value 1.

5. $\mathcal{S}$ receives from $\mathcal{A}$ the encrypted polynomials $\{q_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$.

6. $\mathcal{S}$ receives from $\mathcal{A}$ its input $\{Q_{i,j}, r_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$ for the ideal computation of $\mathcal{R}_{\mathrm{POLY}}$. If the conditions for outputting $(\lambda, 1)$ are not met then $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_\cup$ and aborts.

7. $\mathcal{S}$ sets $\widetilde{X} = \cup_{i=1}^{B} \{x : Q_i(x) = 0 \wedge (h_0(x) = i \vee h_1(x) = i)\}$ and completes $\widetilde{X}$ to size m_X by adding $m_X - |\widetilde{X}|$ arbitrary elements outside of the domain $\{0,1\}^{p(n)}$. That is, $\mathcal{S}$ chooses an arbitrary set $\widehat{X}$ of appropriate size from $\{0,1\}^{p(n)}$ and concatenates the bit one to every element in this set. $\mathcal{S}$ sends $\widetilde{X} \cup \widehat{X}$ to the trusted party for $\mathcal{F}_\cup$ and receives as answer a set $Z = \widetilde{X} \cup \widehat{X} \cup Y$. $\mathcal{S}$ sets $\widetilde{Y}$ to $Z \backslash (\widetilde{X} \cup \widehat{X})$ and completes it to the size m_Y by choosing the same arbitrary element from $\widetilde{X}$ $m_Y - |\widetilde{Y}|$ times.

8. Let $y_1, \ldots, y_{m_Y}$ be a random ordering of the elements of set $\widetilde{Y}$. For $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ considers each commitment $\mathsf{com}_\alpha = h^{\hat{y}_\alpha} g^{\hat{s}_\alpha}$ in conjunction with y_α and performs the following:

   (a) $\mathcal{S}$ computes a value s_α such that $\mathsf{com}_\alpha = h^{y_\alpha} g^{s_\alpha}$.[13]

   (b) $\mathcal{S}$ chooses a random string r_α (of the outcome length of the pseudorandom function) and records the pair (s_α, r_α).

   (c) $\mathcal{S}$ parses r_α as $r_0 \| r_1 \| \hat{r}_0 \| \hat{r}_1 \| r_\oplus \| r_z$ and completes Step 6 of the protocol playing the role of the honest P_2.

9. $\mathcal{S}$ emulates for m_Y times the ideal computation of F_{PRF} as follows: One pair (s_α, r_α) is considered for each emulation of F_{PRF}, in the order in which the pairs were recorded. If it happens that $\mathcal{A}$ enters a value s such that $s = s_\alpha$, then $\mathcal{S}$ returns r_α, otherwise, $\mathcal{S}$ returns a randomly chosen string of the same length.

10. $\mathcal{S}$ completes the execution as the honest P_2 using its set $\widetilde{Y}$. Specifically, $\mathcal{S}$ receives from $\mathcal{A}$ its inputs for the ideal computations of $\mathcal{R}_{\mathrm{COUNT}}$ and $\mathcal{R}_{\mathrm{NZ}}$. If the conditions for outputting $(\lambda, 1)$ are not met then $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_\cup$ and aborts. Finally, if $\mathcal{A}$ does not send $Z \backslash \widehat{X}$ in Step 8 or $|\widetilde{X}| \neq m_X$, $\mathcal{S}$ aborts sending $\perp$ to $\mathcal{F}_\cup$ (we prove below that $\mathcal{S}$, playing the role of the honest P_2, must abort within Step 9 if these conditions are nor met).

11. If the execution is successfully completed, then $\mathcal{S}$ sends output to the trusted party for $\mathcal{F}_\cup$. Otherwise it sends $\perp$.

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

Recall that we compare the simulated execution to a hybrid execution where a trusted party is used to compute the ideal functionality F_{PRF}, and the zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathrm{COUNT}}$, $\mathcal{R}_{\mathrm{POLY}}$, $\mathcal{R}_{\mathrm{COM}}$ and $\mathcal{R}_{\mathrm{DL}}$. To prove that $\mathcal{A}$'s output in the hybrid and simulated executions are computationally close we construct a sequence of hybrid games and show that the corresponding random variables $\mathrm{H}_\ell^{\mathcal{A}(z)}(X, Y, n)$ that consist of the output of $\mathcal{A}$ in hybrid game H_ℓ are computationally close.

Game H_0**:** The simulated execution.

Game H_1**:** The simulator $\mathcal{S}_1$ does not interact with the trusted party for $\mathcal{F}_\cup$ and is given P_2's real input Y instead. $\mathcal{S}_1$ works exactly like $\mathcal{S}$ except that in Step 8 of the simulation it considers the commitments com_α in conjunction with the elements of Y (whereas $\mathcal{S}$ uses $\widetilde{Y}$). In addition, $\mathcal{S}_1$ does not send $\widetilde{X} \cup \widehat{X}$ to the trusted party, but uses Y to compute and output $Y \cap \widetilde{X}$. Finally, $\mathcal{S}_1$ outputs the joint output of $\mathcal{A}$ and P_2 (as computed in this execution). The only difference between these games is that $\mathcal{S}$ uses an arbitrary element from $\widetilde{X}$ to complete $\widetilde{Y}$ to size m_Y (we denote this multi-set by $\widetilde{Y}'$), whereas $\mathcal{S}_1$ uses $Y' = Y \cap \widetilde{X}$.

We claim that the distributions in these two executions are computationally close. Let $y \in Y'$ (resp. $y \in \widetilde{Y}'$) be the α element considered in Game H_1 (resp. Game H_0), then $\mathcal{A}$ receives e_α^0 and e_α^1. Note that e_α^0, e_α^1 are encryptions of random values and hence they do not convey any information about s_α (and hence do not affect the probability of guessing s_α). In particular, the guessing of s_α should only be based on retrieving it from com_α, which, *even if y_α is given explicitly*, requires solving the discrete logarithm problem in $\mathbb{G}$. Similarly, e_z is an encryption of a random value and t_α is a random string, hence no information is revealed about s_α.

Game H_2**:** The simulator $\mathcal{S}_2$ is identical to $\mathcal{S}_1$ except that instead of computing every commitment com_α based on a random value $\hat{y}_\alpha$, and then decommitting it into $y_\alpha \in Y$ it commits to elements of Y from the

[13] Recall that $\mathcal{S}$ knows $t = \log_g h$ and thus is able to decommit com_α into any value of its choice.

start. As com is a perfectly hiding commitment scheme, $\mathrm{H}_1^{\mathcal{A}(z)}(X,Y,n)$ and $\mathrm{H}_2^{\mathcal{A}(z)}(X,Y,n)$ are identically distributed.

Game H_3**:** The simulator $\mathcal{S}_3$ is given oracle access to a truly random function H_{Func} and hence does not evaluate it by itself. This makes no difference for the output distribution – the random variables $\mathrm{H}_2^{\mathcal{A}(z)}(X,Y,n)$ and $\mathrm{H}_3^{\mathcal{A}(z)}(X,Y,n)$ are identically distributed.

Game H_4**:** The simulator $\mathcal{S}_4$ is given oracle access to a pseudorandom function $F_{\mathrm{PRF}}(k,\cdot)$. The computational indistinguishability of $\mathrm{H}_3^{\mathcal{A}(z)}(X,Y,n)$ and $\mathrm{H}_4^{\mathcal{A}(z)}(X,Y,n)$ follows from the pseudorandomness of F_{PRF}.

Game H_5**:** The hybrid execution. The only difference between game H_4 and the hybrid execution is due to the fact that $\mathcal{S}_4$ aborts if $\mathcal{A}$ does not send the set $Z\backslash\widehat{X}$ in Step 8, whereas the hybrid P_2 verifies $\mathcal{A}$'s computations within Step 9. However, recall that according to the checks of P_2, the output that $\mathcal{A}$ sends includes Y and $\widetilde{X}$ and nothing else; see a detailed discussion above. Then, combined with the fact that $\mathcal{A}$ cannot cheat in the ideal computations of $\mathcal{R}_{\mathrm{COUNT}}$ and $\mathcal{R}_{\mathrm{NZ}}$ we conclude that both $\mathcal{S}_4$ and the honest P_2 outputs the same set.

***P_2* is corrupted.** Let $\mathcal{A}$ denote an adversary controlling P_2 we construct a simulator $\mathcal{S}$ as follows. We construct a simulator $\mathcal{S}$ that sends zero polynomials instead of B polynomials that were computed based on the real input X. Furthermore, $\mathcal{S}$ does not use the secret key during the execution, but uses the values it extracts from the execution of $\mathcal{R}_{\mathsf{com}}$ to recompute $e_\alpha^0, e_\alpha^1, e_z, t_\alpha$. A more formal description follows:

1. $\mathcal{S}$ is given Y, m_X and 1^n and invokes $\mathcal{A}$ on these inputs. $\mathcal{S}$ sets $m_Y = |Y|$.
2. $\mathcal{S}$ chooses $t, t' \leftarrow_R \mathbb{Z}_q$ and sends $\mathcal{A}$ the keys $h = g^t, h' = g^{t'}$. It then emulates the trusted party that computes $\mathcal{R}_{\mathrm{DL}}$ and sends $\mathcal{A}$ the value 1.
3. $\mathcal{S}$ computes the parameters B, M for the balanced allocation scheme and chooses random seeds for the hash functions h_0, h_1. These are then sent to $\mathcal{A}$.
4. $\mathcal{S}$ receives from $\mathcal{A}$ a set of m_Y commitments $\{\mathsf{com}_\alpha\}_{\alpha=1}^{m_Y}$, supposedly of elements in Y. For each commitment com_α that it receives, $\mathcal{S}$ emulates the trusted party that computes $\mathcal{R}_{\mathsf{com}}$. It receives $\mathcal{A}$'s input for the ideal computation of $\mathcal{R}_{\mathsf{com}}$, which is a pair (y_α, s_α). If $(\mathsf{com}_\alpha, (y_\alpha, s_\alpha)) \notin \mathcal{R}_{\mathsf{com}}$ then $\mathcal{S}$ aborts, sending $\perp$ to the ideal functionality for $\mathcal{F}_\cup$.
5. $\mathcal{S}$ sends to $\mathcal{A}$ the encryptions of B zero polynomials.
6. $\mathcal{S}$ emulates the trusted party for $\mathcal{R}_{\mathrm{POLY}}$. It receives from $\mathcal{A}$ a set of BM coefficients and pk. If $\mathcal{A}$'s input is the exact set of encryptions that it received from $\mathcal{S}$ in the previous step, and pk then $\mathcal{S}$ returns 1, otherwise it returns 0.
7. for each $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ receives e_α^0, e_α^1 and t_α from $\mathcal{A}$ and engage in an execution of oblivious selection with $\mathcal{A}$, where it sets z_0, z_1 to zero encryptions. $\mathcal{S}$ emulates the ideal execution for $\mathcal{R}_{\mathrm{COUNT}}$ and receives an encryption e_z.
8. Recall that for each $e_\alpha^0, e_\alpha^1, t_\alpha$, $\mathcal{S}$ recorded in Step 4 of the simulation $\mathcal{A}$'s input (y_α, s_α) to $\mathcal{R}_{\mathsf{com}}$. $\mathcal{S}$ now receives $\mathcal{A}$'s input k for the ideal computation F_{PRF} corresponding to y_α,[14] and evaluates $r_0\|r_1\|\hat{r}_0\|\hat{r}_1\|r_\oplus\|r_z = F_{\mathrm{PRF}}(k, s_\alpha)$. It then checks if $e_\alpha^0, e_\alpha^1, e_z, t_\alpha$, are consistent with $r_0, r_1, \hat{r}_0, \hat{r}_1, r_\oplus, r_z$

[14] note that $\mathcal{A}$ may feed $\mathcal{S}$ with a different k in every round of this loop.

when P_2's input is y_α. That is, $\mathcal{S}$ recomputes these encryptions using $s_\alpha, F_{\mathrm{PRF}}(k, s_\alpha)$ and y_α as the honest P_2 would in the real execution, and checks whether the result equals $e^0_\alpha, e^1_\alpha, e_z, t_\alpha$. If the check succeeds, $\mathcal{S}$ locally records the value y_α.

9. $\mathcal{S}$ sets $\widetilde{Y}$ to the set of recorded values, and completes $\widetilde{Y}$ to the size m_Y by adding $m_Y - |\widetilde{Y}|$ arbitrary elements outside of the domain $\{0,1\}^{p(n)}$. That is, $\mathcal{S}$ chooses an arbitrary set $\widehat{Y}$ of appropriate size from $\{0,1\}^{p(n)}$ and concatenates the bit one to every element in this set.

10. $\mathcal{S}$ sends $\widetilde{Y} \cup \widehat{Y}$ to the trusted party and receives as answer a set $Z = X \cup \widetilde{Y} \cup \widehat{Y}$. $\mathcal{S}$ sets $Y' = Z \backslash \widehat{Y}$ and sends $\mathcal{A}$ this set.

11. $\mathcal{S}$ completes the execution by emulating the ideal executions for $\mathcal{R}_{\mathrm{COUNT}}$ and $\mathcal{R}_{\mathrm{NZ}}$. If the execution is successfully completed, then $\mathcal{S}$ sends output to the trusted party for $\mathcal{F}_\cup$. Otherwise it sends $\perp$.

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

We note that there are two differences between the simulated and the hybrid executions. First, $\mathcal{S}$ sends the encryptions of B zero polynomials instead of B polynomials that were computed based on the real input X, and second, $\mathcal{S}$ does not use sk during its execution, and instead uses the information $\mathcal{A}$ sends as input to $\mathcal{R}_{\mathsf{com}}$. In the following we define a sequence of hybrid games and denote by the random variable $\mathrm{H}^{\mathcal{A}(z)}_\ell(X, Y, n)$ (for a fixed n) the joint output of $\mathcal{A}$ and P_1 in hybrid game H_ℓ.

Game H_0**:** The simulated execution.

Game H_1**:** The simulator $\mathcal{S}_1$ acts identically to $\mathcal{S}$ except that it does not get to know the secret keys $t = \log_g h$ and $t' = \log_g h'$. The random variables $\mathrm{H}^{\mathcal{A}(z)}_0(X, Y, n)$ and $\mathrm{H}^{\mathcal{A}(z)}_1(X, Y, n)$ are identically distributed as both $\mathcal{S}$ and $\mathcal{S}_1$ do not use t, t'.

Game H_2**:** In this game there is no trusted party and no honest P_1. Instead, the simulator $\mathcal{S}_2$ is given as input P_1's real input X. $\mathcal{S}_2$ works exactly like $\mathcal{S}_1$, except that instead of sending arbitrary polynomials, it computes the polynomials as in the hybrid execution using the set X. In addition, $\mathcal{S}_2$ does not send $\widetilde{Y} \cup \widehat{Y}$ to the trusted party, but uses X to compute and output $X \cap \widetilde{Y}$. The proof that $\mathrm{H}^{\mathcal{A}(z)}_1(X, Y, n)$ and $\mathrm{H}^{\mathcal{A}(z)}_2(X, Y, n)$ are computationally indistinguishable is by reduction to the semantic security of the El Gamal encryption scheme.

Game H_3**:** The simulator $\mathcal{S}_3$ acts identically to $\mathcal{S}_2$ except that $\mathcal{S}_3$ is given $t' = \log_g h'$ (but not $t = \log_g h$). The random variables $\mathrm{H}^{\mathcal{A}(z)}_2(X, Y, n)$ and $\mathrm{H}^{\mathcal{A}(z)}_3(X, Y, n)$ are identically distributed as $\mathcal{S}_2$ and $\mathcal{S}_3$ do not use t'.

Game H_4**:** $\mathcal{S}_4$ computes the union as in the hybrid execution. To conclude the proof, We show that $\mathrm{H}^{\mathcal{A}(z)}_3(X, Y, n)$ and $\mathrm{H}^{\mathcal{A}(z)}_4(X, Y, n)$ are statistically close. Note that $\mathcal{S}_3, \mathcal{S}_4$ act identically until they get to the computation of the intersection set, where:

- In Game H_3, the simulator $\mathcal{S}_3$ extracts pairs (y_α, s_α) from the zero-knowledge proof of knowledge for $\mathcal{R}_{\mathsf{com}}$ on a commitment com_α. For each such pair it later receives a key k for F_{PRF} and values $e^0_\alpha, e^1_\alpha, e_z, t_\alpha$, and adds y_α to $\widetilde{Y}$ if these values are consistent with the randomness $r_0, r_1, \hat{r}_0, \hat{r}_1, r_\oplus, r_z$ derived from $F_{\mathrm{PRF}}(k, s_\alpha)$. The output is computed as $X \cup \widetilde{Y}$.

- In Game H_4, the simulator $\mathcal{S}_4$ follows the same procedure as in the hybrid execution, i.e., That is, $\mathcal{S}_4$ uses sk to decrypt e^0_α, e^1_α and e_z. It then derives s'_α, y'_α and checks that $e^0_\alpha, e^1_\alpha, e_z, t_\alpha$ are consistent with y'_α and randomness $r_0, r_1, \hat{r}_0, \hat{r}_1, r_\oplus, r_z$ derived from $F_{\mathrm{PRF}}(k, s'_\alpha)$.

Observe first that if an element y_α satisfies the conditions for being included in the input in Game H_3, it also satisfies the conditions for being included in the input in Game H_4: if $\mathcal{S}_3$ outputs an element y_α then it must be that $\mathsf{com}_\alpha, e^0_\alpha, e^1_\alpha, e_z, t_\alpha$ were computed according to the protocol specification, and hence $\mathcal{S}_4$ would have outputted it.

Consider now the reverse direction. Let bad denote the event where there exists an element $x \in X$ that $\mathcal{S}_4$ decided to output, but should not have been outputted by $\mathcal{S}_3$. We show that $\Pr[\mathsf{bad}]$ is negligible. Recall that both $\mathcal{S}_3$ and $\mathcal{A}$ do not know $\log_g h$ and that $\mathcal{S}_3$ knows the secret key sk. Note that for bad to occur it must be that for some commitment com_α, (i) $\mathcal{A}$ gives a pair (y_α, s_α) as input to the functionality $\mathcal{R}_{\mathsf{com}}$ (this pair is recorded by $\mathcal{S}_4$), and (ii) one of e^0_α, e^1_α is decrypted to $(s')^2 \bmod q'$ where $(s')^2 \bmod q' \neq (s_\alpha)^2 \bmod q'$ such that for some $x \in X$ the values e^0_α, e^1_α are consistent with setting $y = x$ and the randomness obtained from $F_{\mathrm{PRF}}(k, s')$, and furthermore $\mathsf{com}_\alpha = g^x h^{s'}$. We can therefore construct a non-uniform algorithm that given $g, h \leftarrow_R \mathbb{Z}_q$ for prime q, succeeds in computing $y_\alpha, s_\alpha, x, s'$ such that $g^{y_\alpha} h^{s_\alpha} = g^x h^{s'}$, or equivalently succeeds in computing $\log_g h$.

■

4.2.1 Efficiency

The analysis is as in Protocol 5. We First note that the protocol is constant round. The costs of using current implementations of F_{PRF} on inputs of length $p(n)$ is that of $p(n)$ oblivious transfer invocations, and hence of $O(p(n))$ modular exponentiations. We get that the overall communication costs are of sending $O(m_X p(n) + m_Y)$ group elements, and the computation costs are of performing $O(m_X p(n) + m_Y \log\log n)$ modular exponentiations.

5 Security Under Universal Composability

Our protocols can be transformed to protocols that are secure in the UC (universal composability) framework. We show how to modify protocols $\pi_\cap$ and $\pi_\cup$ into protocols $\pi^{\mathrm{UC}}_\cap$ and $\pi^{\mathrm{UC}}_\cup$ that compute the respective functionalities $\mathcal{F}_\cap$ and $\mathcal{F}_\cup$ with security under universal composability [9].

A protocol that is universally composable maintains its security even when run in an arbitrary network setting concurrently with other secure and insecure protocols [9]. The formal definition introduces an additional adversarial entity $\mathcal{Z}$, called the *environment*. The environment generates the inputs to all parties, receives all outputs, and interacts with the adversary in an arbitrary way throughout the computation. A protocol π securely computes an ideal functionality $\mathcal{F}$ in the UC framework for any adversary $\mathcal{A}$ that interacts with the parties running the protocol, there exists an ideal process adversary (or "simulator") $\mathcal{S}$ that interacts with the trusted party, so that no environment $\mathcal{Z}$ can distinguish the two cases. If the above holds for a protocol π we say that π *UC realizes* $\mathcal{F}$ and that π is *UC secure*; see [9] for a formal definition and the proof that the definition implies security under composition in an arbitrary network as described above.

We remark that secure two-party computation of most interesting functionalities in this model requires an additional trusted setup assumption such as a common reference string (CRS) [10]. Our protocols of UC secure protocols employ sub-protocols that use a common reference string.

We begin with the observation that a security proof with straight-line simulators is essentially enough for proving UC security. We thus identify all the locations in our proofs in which the simulator uses rewinding and handle them individually. As most of these rewindings are due to simulating zero-knowledge Σ-protocols, we first construct a general proof that UC realizes every Σ-protocol.[15] This, in turn, enables to construct non-rewinding simulators for $\pi_\cap$ and $\pi_\cup$.

[15] A Σ-protocol is a three rounds proof in a format of commitment, challenge, response. See [17] for more details.

5.1 UC Zero-Knowledge for Σ-Protocols

For completeness, we present a general UC construction for Σ-protocols (that are three-round) which combines the techniques of [14, 26]. We first have the verifier commit to its challenge followed by the prover who commits to its first message. Both parties use a UC commitment scheme $(\mathsf{com}_{\mathrm{UC}}, \mathsf{dec}_{\mathrm{UC}})$ that UC realizes functionality $\mathcal{F}_{\mathrm{COM}}$ (Figure 3 recalls the definition of [17]). The protocol of [17] realizes $\mathcal{F}_{\mathrm{COM}}$ and is secure assuming the hardness of the *decisional composite residuosity problem* in the common reference string model, and operates in $\mathbb{Z}_N$.

Functionality $\mathcal{F}_{\mathrm{COM}}$

Functionality $\mathcal{F}_{\mathrm{COM}}$ proceeds as follows, running with parties P_1 and P_2 and an adversary $\mathcal{S}$.

- Generate a uniformly random modulus N that is a product of two odd primes p and q. Send N to parties P_1 and P_2, and send p, q to $\mathcal{S}$.
- Upon receiving a message $(\mathsf{commit}, id, P_i, P_j, m)$ from P_i, send $(\mathsf{receipt}, id, P_i, P_j)$ to P_j and $\mathcal{S}$. Record (id, P_i, P_j, m) and ignore subsequent messages of the form $(\mathsf{commit}, id, \ldots)$.
- Upon receiving a message $(\mathsf{open}, id, P_i, P_j)$ from P_i where (id, P_i, P_j, m) is recorded, send $(\mathsf{open}, id, P_i, P_j, m)$ to P_j and to $\mathcal{S}$. Otherwise, do nothing.

Figure 3: The commitment functionality [17]

Next we have the prover choose two distinct challenges and commit to their corresponding replies (meaning, the prover computes the response in both cases and commits to these values). Finally, the verifier reveals its challenge and the prover opens the commitment that corresponds to this bit.[16] This concludes the high level construction of our protocol that UC realizes the zero-knowledge functionality $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$, parameterized by a relation $\mathcal{R}$; see Figure 4.

Functionality $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$

- Upon receiving a message $(\mathsf{ZK-prover}, id, x, \omega)$ from P, do: if $\mathcal{R}(x, \omega) = 1$, then send $(\mathsf{ZK-proof}, id, x)$ to V and $\mathcal{S}$ and halt. Otherwise, halt.

Figure 4: The zero-knowledge functionality

Theorem 5.1 *Let π denotes a Σ-protocol for a binary relation $\mathcal{R}$. Then under the hardness assumption of the decisional composite residuosity problem, there exists a protocol $\pi_{\mathcal{R}}$ that UC realizes $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$ in the $\mathcal{F}_{\mathsf{com}}$-hybrid model with soundness half.*

Proof: Let $\pi_{\mathcal{R}}$ denotes a Σ-protocol for a relation $\mathcal{R}$, we present our protocol $\pi_{\mathcal{R}}^{\mathrm{UC}}$.

Protocol 9 (UC zero-knowledge for a relation $\mathcal{R}$):

- **Joint statement:** *x and a security parameter* 1^n.
- **Auxiliary input for the prover:** *A witness ω.*

[16] Two distinct accepting transcripts enable to extract the witness in all the proofs included in this section. This technique can be extended for any polynomial number of distinct transcripts needed for extraction.

- **The protocol:**
  1. *The verifier chooses a random bit $b \in \{0,1\}$ and sends $\mathcal{F}_{\mathsf{com}}$ the message $(\mathsf{commit}, id, V, P, b)$.*
  2. *The prover P computes α as in the original Σ-protocol, $\pi_{\mathcal{R}}$. It then chooses two distinct challenges β_0 and β_1, and computes the responses γ_0 and γ_1 to these challenges.*
  3. *P sends the verifier the values $\{\alpha, \beta_i\}_{i\in\{0,1\}}$. Furthermore, it sends $\mathcal{F}_{\mathsf{com}}$ the messages $(\mathsf{commit}, id_0, P, V, \gamma_0)$ and $(\mathsf{commit}, id_1, P, V, \gamma_1)$.*
  4. *If $\beta_0 = \beta_1$, V aborts. Otherwise, it responds P by sending the message $(\mathsf{open}, id, V, P)$ to $\mathcal{R}_{\mathsf{com}}$.*
  5. *Upon receiving the message $(\mathsf{open}, id, V, P, b)$ from $\mathcal{F}_{\mathsf{com}}$, P returns it the message $(\mathsf{open}, id_b, P, V)$.*
  6. *Upon receiving the message $(\mathsf{open}, id_b, P, V, \gamma_b)$ from $\mathcal{F}_{\mathsf{com}}$, V accepts only if $(\alpha, \beta_b, \gamma_b)$ is an accepting transcript.*

Proposition 5.1 *Protocol 9 UC realizes $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model with soundness half.*

Proof: Note first that the protocol is correct since in case both parties are honest, P always convinces V by producing a pair of accepting transcripts. We further prove that the prover can only cheat with probability half. This is due to the fact that if it does not know ω it will not be able to commit to two different accepting transcripts, and thus will be caught with probability half. Let $\mathcal{A}$ be an adversary that interacts with the prover P and the verifier V running Protocol 9. We construct an adversary $\mathcal{S}$ for the ideal process for $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and Protocol 9 or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$. Recall that $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$ and with the environment $\mathcal{Z}$. Simulator $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with $\mathcal{Z}$ and parties P and V. $\mathcal{S}$ proceeds as follows:

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape (to be read by $\mathcal{S}$'s environment $\mathcal{Z}$).

Simulating the case where P is corrupted:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{com}}$ and internally sends $\mathcal{A}$ the message $(\mathsf{receipt}, id, V, P)$.
2. $\mathcal{S}$ internally receives from $\mathcal{A}$ two sets $\{\alpha, \beta_i\}_{i\in\{0,1\}}$ and its messages to $\mathcal{F}_{\mathsf{com}}$; $(\mathsf{commit}, id_0, P, V, \gamma_0)$ and $(\mathsf{commit}, id_1, P, V, \gamma_1)$ and aborts in case $\beta_0 = \beta_1$.
3. If $(\alpha, \beta_0, \gamma_0)$ and $(\alpha, \beta_1, \gamma_1)$ denote a pair of valid distinct transcripts $\mathcal{S}$ computes ω. Otherwise $\mathcal{S}$ halts.
4. $\mathcal{S}$ completes the simulation by sending $\mathcal{A}$ the message $(\mathsf{open}, id, V, P, b)$ for a random bit b, and verifies that $\mathcal{A}$ sends $\mathcal{F}_{\mathsf{com}}$ the message $(\mathsf{open}, id_b, P, V, \gamma_b)$. In this case $\mathcal{S}$ sends (x, ω) to the trusted party for $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}$. Otherwise it halts.

Note that $\mathcal{A}$'s view is identical in both the real and the simulated executions as all the messages from V are via $\mathcal{F}_{\mathsf{com}}$ and b is uniform in both executions. Furthermore, the simulation fails with probability at most $\frac{1}{2}$, since if the honest verifier accepts the proof with probability that is greater than $\frac{1}{2}$, then it must be that the corrupted prover committed to two distinct valid transcripts which enable the simulator to extract ω.

Simulating the case where V is corrupted:

1. Upon receiving a message $(\mathsf{commit}, id, V, P, b)$, $\mathcal{S}$ chooses two distinct challenges β_0 and β_1 as the honest prover does, and computes α as in the simulation of $\pi_{\mathcal{R}}$. That is, $\mathcal{S}$ invokes the simulator $\mathcal{S}_{\mathcal{R}}$ of $\pi_{\mathcal{R}}$ and plays the role of the honest verifier who sends the challenge β_b. Let α be the first message in the transcript that $\mathcal{S}_{\mathcal{R}}$ outputs. Then $\mathcal{S}$ internally sends $\mathcal{A}$ the sets $\{\alpha, \beta_i\}_{i\in\{0,1\}}$. In addition, $\mathcal{S}$ sends V the messages $(\mathsf{receipt}, id_0, P, V)$ and $(\mathsf{receipt}, id_1, P, V)$.

2. Upon receiving a message $(\mathsf{open}, id, V, P)$, $\mathcal{S}$ internally sends $\mathcal{A}$ the message $(\mathsf{open}, id_b, P, V, \gamma_b)$.

We claim that $\mathcal{A}$'s view in the hybrid and the simulated executions is identical. This is due to the fact that the transcript $(\alpha, \beta_b, \gamma_b)$ is identically distributed in both executions based on the zero-knowledge property of $\pi_{\mathcal{R}}$.

Simulating the case that neither party is corrupted: In this case, $\mathcal{S}$ generates a simulated transcript of messages between the hybrid model parties as in the above simulations. We further claim that the simulated and the hybrid transcripts are identical in the presence of an eavesdropper $\mathcal{A}$ following the same claims as above. Then, combined with the correctness argument we conclude that the joint output distribution is identical in both executions.

Simulating the case that both parties are corrupted: The simulator $\mathcal{S}$ just runs $\mathcal{A}$ internally who generates the messages from both P and V by itself. ■
This concludes our proof. ■

Soundness reduction. Seeing that the soundness for this UC proof is only half, which is inadequate for our purposes, we will need to repeat it enough times to achieve a negligible soundness. That is, in order to achieve negligible soundness in a statistical parameter 1^s we invoke s independent copies of this proof in parallel; see [26] for more details.

5.2 UC Set-Intersection

We now discuss how to construct a new protocol $\pi_{\cap}^{\mathrm{UC}}$ that UC realizes functionality $\mathcal{F}_{\cap}$. Similar modifications apply to the set-union protocol. Observe that the simulator for $\pi_{\cap}$ uses rewinding within the zero-knowledge proofs and the subprotocol π_{PRF}. We replace the standard zero-knowledge proofs, the commitment scheme $(\mathsf{com}, \mathsf{dec})$, and protocol π_{PRF} with UC constructions (protocol $\pi_{\mathrm{PRF}}^{\mathrm{UC}}$ is a modified version of π_{PRF} [23] where every oblivious transfer invocation is replaced with a UC oblivious transfer, we omit the detail here). Recall that π_{POLY} is not a standard Σ proof, rather it employs such proofs. It can be modified into a UC proof by employing UC subprotocols. We denote by $\pi_{\mathrm{DL}}^{\mathrm{UC}}$, $\pi_{\mathrm{POLY}}^{\mathrm{UC}}$ and $\pi_{\mathrm{PRF}}^{\mathrm{UC}}$ the UC constructions for the respective functionalities $\mathcal{F}_{\mathrm{ZK}}^{\mathrm{DL}}$, $\mathcal{F}_{\mathrm{POLY}}$ and $\mathcal{F}_{\mathrm{PRF}}$, and conclude with the following statement,

Theorem 5.2 *Assume that $\pi_{\mathrm{DL}}^{\mathrm{UC}}$, $\pi_{\mathrm{POLY}}^{\mathrm{UC}}$ and $\pi_{\mathrm{PRF}}^{\mathrm{UC}}$ are as described above, that (G, E, D) is the El Gamal encryption scheme, and that $\mathsf{com}_{\mathrm{UC}}$ is a UC commitment scheme. Then $\pi_{\cap}^{\mathrm{UC}}$ UC realizes $\mathcal{F}_{\cap}^{\mathrm{UC}}$ in the $\{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}}, \mathcal{F}_{\mathrm{PRF}}\}$-hybrid model in the presence of malicious adversaries.*

Our proof is not modular in $\mathcal{F}_{\mathsf{com}}$ since we rely on the fact that P_1 is given the commitment and not just a receipt as in the ideal setting. In particular, it is essential that P_2 chooses the randomness for $\mathsf{com}_{\mathrm{UC}}$ by itself. We continue by briefly describing the construction of [17]. Let N be an RSA modulus. The hardness assumption of the decisional composite residuosity problem asserts that no probabilistic polynomial adversary can distinguish between $(1+N)^m \cdot r^N \bmod N^2$ and $r^N \bmod N^2$, where $r \leftarrow_R \mathbb{Z}_N^*$ and m is an arbitrary

value in $\mathbb{Z}_N$.[17] Then the common reference string for the above construction includes an RSA modulus N, and a public-key pk which guarantees the following. If it is of the form $(1+N)^m \cdot r^N \bmod N^2$ then the scheme is prefect binding, where a commitment c is computed by $\mathsf{com}_{\mathrm{UC}}(m';r') = pk^{m'} \cdot r'^N \bmod N^2$. An extraction of the decommitted value can be done efficiently using the factorization of N. Otherwise, it is prefect hiding and equivocal. For completeness we present the $\mathcal{F}_{\mathrm{CRS}}$ functionality in Figure 5.

Functionality $\mathcal{F}_{\mathrm{CRS}}$

Functionality $\mathcal{F}_{\mathrm{CRS}}$ runs with parties P_1 and P_2 and is parameterized by a distribution D.

- When activated for the first time on input (id, P_i, P_j) from P_i, choose a value $crs \leftarrow_R D$, send crs back to P_i, and send (crs, P_i, P_j) to the adversary. Next, when receiving (id, P_i, P_j) from P_j (and only P_j), send (id, crs) to P_j and to the adversary, and halt.

Figure 5: The common reference string functionality

Proof: Let $\mathcal{A}$ be an adversary that interacts with parties P_1 and P_2 running $\pi_{\cap}^{\mathrm{UC}}$. We construct an adversary $\mathcal{S}$ for the ideal process for $\mathcal{F}_{\cap}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and $\pi_{\cap}^{\mathrm{UC}}$ or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\cap}$. Recall that $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\cap}$ and with the environment $\mathcal{Z}$. Simulator $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with $\mathcal{Z}$ and parties P_1 and P_2. In the following proof, we only present the constructions of the simulators, as the proof follows similarly to the proof of Theorem 3.3 with the exception that now we omit the games that are related to the commitment scheme com. Specifically, in case P_1 is corrupted games H_1 and H_2 are omitted. $\mathcal{S}$ proceeds as follows:

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape (to be read by $\mathcal{S}$'s environment $\mathcal{Z}$).

Simulating the case where party P_1 is corrupted:

1. Whenever $\mathcal{S}$ internally receives from $\mathcal{A}$, $(\mathsf{ZK}-\mathsf{prover}, id, (\mathbb{G}, g, q, h), t)$ for the ideal computations of $\mathcal{F}_{\mathrm{ZK}}^{\mathrm{DL}}$, it records t and internally passes $\mathcal{A}$ the message $(\mathsf{ZK}-\mathsf{proof}, id, (\mathbb{G}, g, q, h))$ only if $h = g^t$. Otherwise it halts.

2. $\mathcal{S}$ internally receives from $\mathcal{A}$ the parameters B, M and the seeds for the two (pseudo-)random hash functions $h_0, h_1 : \{0,1\}^{p(n)} \to [B]$ used in the balanced allocation scheme. If the parameters B, M were not computed correctly, $\mathcal{S}$ halts.

3. Upon receiving a message (id, P_1, P_2), $\mathcal{S}$ chooses a value crs as follows. It first chooses two odd primes p and q of appropriate length and sets $N = p \cdot q$. It then sets $pk = r^N \bmod N^2$ for a random $r \in \mathbb{Z}_N^*$. $\mathcal{S}$ sets $crs = (N, pk)$, and internally hands $\mathcal{A}$ the message (crs, P_1, P_2).

4. For all $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ chooses $(\hat{y}_\alpha, \hat{s}_\alpha) \leftarrow_R \mathbb{Z}_q \times \mathbb{Z}_q$ and internally sends $\mathcal{A}$ the commitment $\mathsf{com}_{\mathrm{UC}}^{\alpha}(\hat{y}_\alpha; \hat{s}_\alpha)$ under pk.

[17] This assumption was generalized by [16].

5. $\mathcal{S}$ internally receives from $\mathcal{A}$ the message,

   $(\mathsf{ZK}-\mathsf{prover}, id, \{Q_{i,j}, r_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}, (\{q_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}, m_X, pk))$ for the ideal computation of $\mathcal{F}_{\mathrm{ZK}}^{\mathrm{POLY}}$. If the conditions for outputting $(\lambda, 1)$ are not met then $\mathcal{S}$ halts.

6. $\mathcal{S}$ sets $\widetilde{X} = \cup_{i=1}^{B}\{x : Q_i(x) = 0 \wedge (h_0(x) = i \vee h_1(x) = i)\}$ and completes $\widetilde{X}$ to size m_X by adding random elements from $\{0,1\}^{p(n)}$. $\mathcal{S}$ sends $\widetilde{X}$ to the trusted party for $\mathcal{F}_{\cap}$ and receives as answer a set $Z = \widetilde{X} \cap Y$. $\mathcal{S}$ sets $\widetilde{Y}$ to Z and completes $\widetilde{Y}$ to the size m_Y by adding random elements from $\{0,1\}^{p(n)}$.

7. Let $y_1, \ldots, y_{m_Y}$ be a random ordering of the elements of set $\widetilde{Y}$. For $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ considers each commitment $\mathsf{com}_{\mathrm{UC}}^{\alpha}$ in conjunction with y_α and performs the following:

   (a) $\mathcal{S}$ computes a value s_α such that $\mathsf{com}_{\mathrm{UC}}^{\alpha} = \mathsf{com}(y_\alpha; s_\alpha)$.

   (b) $\mathcal{S}$ chooses a random string r_α (of the outcome length of the pseudorandom function) and records the pair (s_α, r_α).

   (c) $\mathcal{S}$ parses r_α as $r_0\|r_1\|\hat{r}_0\|\hat{r}_1$ and completes Step 6 of the protocol playing the role of the honest P_2.

8. $\mathcal{S}$ internally emulates the ideal computation of $\mathcal{F}_{\mathrm{PRF}}$ as follows: One pair (s_α, r_α) is considered for each oblivious PRF execution, in the order in which the pairs were recorded. If it happens that $\mathcal{A}$ enters a value s such that $s = s_\alpha$, then $\mathcal{S}$ returns r_α, otherwise, $\mathcal{S}$ returns a randomly chosen string of the same length.

Simulating the case where party P_2 is corrupted:

1. $\mathcal{S}$ chooses $t \leftarrow_R \mathbb{Z}_q$ and internally sends P_2 the keys $h = g^t$. It then emulates functionality $\mathcal{R}_{\mathrm{ZK}}^{\mathrm{DL}}$ and internally sends $\mathcal{A}$ the message $(\mathsf{ZK}-\mathsf{proof}, id, (\mathbb{G}, g, q, h))$.

2. $\mathcal{S}$ computes the parameters B, M for the balanced allocation scheme and chooses random seeds for the hash functions h_0, h_1. These are then internally sent to $\mathcal{A}$.

3. Upon receiving a message (id, P_1, P_2) from $\mathcal{A}$, $\mathcal{S}$ chooses a value crs as follows. It first chooses two odd primes p and q of appropriate length and sets $N = p \cdot q$. It then sets $pk = (1+N)^m \cdot r^N \bmod N^2$ for an arbitrary $m \in \mathbb{Z}_N$ and a random $r \in \mathbb{Z}_N^*$. $\mathcal{S}$ sets $crs = (N, pk)$, and internally hands $\mathcal{A}$ the message (id, crs).

4. $\mathcal{S}$ receives from $\mathcal{A}$ a set of m_Y commitments $\{\mathsf{com}_{\mathrm{UC}}^{\alpha}\}_{\alpha=1}^{m_Y}$, supposedly of elements in Y. For each commitment $\mathsf{com}_{\mathrm{UC}}^{\alpha}$ that it receives, $\mathcal{S}$ computes (y_α, s_α) such that $\mathsf{com}_{\mathrm{UC}}^{\alpha} = (y_\alpha; s_\alpha)$ using the knowledge of p and q.

5. $\mathcal{S}$ sends to $\mathcal{A}$ the encryptions of B zero polynomials.

6. $\mathcal{S}$ emulates the trusted party for $\mathcal{F}_{\mathrm{ZK}}^{\mathrm{POLY}}$ and internally sends $\mathcal{A}$ the message

   $(\mathsf{ZK}-\mathsf{proof}, id, (\{q_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}, m_X, pk))$.

7. for each $\alpha \in \{1, \ldots, m_Y\}$, $\mathcal{S}$ receives e_α^0, e_α^1 from $\mathcal{A}$.

8. $\mathcal{S}$ receives $\mathcal{A}$'s input for the ideal computation $\mathcal{F}_{\text{PRF}}$; (id, k) and evaluates $r_0 \| r_1 \| \hat{r}_0 \| \hat{r}_1 = F_{\text{PRF}}(k, s_\alpha)$ for every α. It then checks if e^0_α, e^1_α, are consistent with $r_0, r_1, \hat{r}_0, \hat{r}_1$ when P_2's input is y_α. That is, $\mathcal{S}$ recomputes these encryptions using $s_\alpha, F_{\text{PRF}}(k, s_\alpha)$ and y_α as the honest P_2 would in the real execution, and checks whether the result equals e^0_α, e^1_α. If the check succeeds, $\mathcal{S}$ locally records the value y_α.

9. $\mathcal{S}$ sets $\widetilde{Y}$ to the set of recorded values, and completes $\widetilde{Y}$ to the size m_Y by adding random elements from $\{0,1\}^{p(n)}$.

10. $\mathcal{S}$ sends $\widetilde{Y}$ to the trusted party for $\mathcal{F}_\cap$.

Simulating the case that neither party is corrupted: In this case, $\mathcal{S}$ generates a simulated transcript of messages between the hybrid model parties as in the above simulations. We further claim that the simulated and the hybrid transcripts are identical in the presence of an eavesdropper $\mathcal{A}$ following the same claims as above. Then, combined with the correctness argument we conclude that the joint output distribution is identical in both executions.

Simulating the case that both parties are corrupted: The simulator $\mathcal{S}$ just runs $\mathcal{A}$ internally who generates the messages from both P_1 and P_2 by itself.

■

References

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure Computation of the kth-Ranked Element. In *EUROCRYPT'04*, Springer-Verlag (LNCS 3027), pages 40–55, 2004.

[2] Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *4th TCC*, Springer-Verlag (LNCS 4392), 137–156, 2007.

[3] Y. Azar, A.Z. Broder, A.R. Karlin and E. Upfal. Balanced Allocations. In *SIAM Journal on Computing,* 29(1):180–200, 1999.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non cryptographic fault tolerant distributed computations. In *20th STOC*, pages 1-10, 1988.

[5] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *2nd TCC*, Springer-Verlag (LNCS 3378), pages 325–341, 2005.

[6] S. Böttcher and S. Obermeier. Secure Set Union and Bag Union Computation for Guaranteeing Anonymity of Distrustful Participants. In *Journal of Software* 3(1): pages 9–17, 2008.

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

[8] J. Camenisch, G. M. Zaverucha. Private Intersection of Certified Sets. In *Financial Crypto*, pages 128–127, 2009.

[9] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*, pages 136–145, 2001.

[10] R. Canetti, E. Kushilevitz and Y. Lindell. On the limitations of Universal Composable Two-Party computation Without Set-Up Assumptions. In *44th FOCS*, pages 136–145, 2003.

[11] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th STOC*, pages 11-19, 1988.

[12] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO'94*, Springer-Verlag (LNCS 839), pages 174–187, 1994.

[13] D. Chaum, and T. P. Pedersen. Wallet Databases with Observers. In *CRYPTO'92*, Springer-Verlag (LNCS 740), pages 89–105, 1992.

[14] I. Damgård. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In *EUROCRYPT'00*, Springer-Verlag (LNCS 1807), pages 418–430, 2000.

[15] I. Damgård and M. Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography*, Sprniger-Verlag (LNCS 1992), pages 119–136, 2001.

[16] I. Damgård, M. Jurik and J.B. Nielsen. A Generalization of Paillier's Public-Key System with Applications to Electronic Voting. In *Public Key Cryptography*, 119–136, 2001.

[17] I. Damgård, and J. B. Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. In *CRYPTO'02*, Springer-Verlag (LNCS 2442), pages 3–42, 2002.

[18] D. Dachman-Soled, T. Malkin, M. Raykova and M. Yung. Efficient Robust Private Set Intersection. In *ANCS*, Springer-Verlag (LNCS 5479), pages 125–142, 2009.

[19] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO'84*, Springer-Verlag (LNCS 196), pages 10–18, 1984.

[20] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. In *ACM Transactions on Algorithms (TALG)*, 2(3): 435–472, 2006.

[21] P. Fouque and D. Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In *Asiacrypt*, pages 573–84, 2000.

[22] P. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting of lotteries. In *Financial Crypto*, pages 90–104, 2009.

[23] M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *2nd TCC*, Springer-Verlag (LNCS 3378), pages 303–324, 2005.

[24] M. Freedman, K. Nissim and B. Pinkas. Efficient Private Matching and Set-Intersection. In *EUROCRYPT'04*, Springer-Verlag (LNCS 3027), pages 1–19, 2004.

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

[26] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[27] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *19th STOC*, pages 218-229, 1987.

[28] C. Hazay and Y. Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *5th TCC*, Springer-Verlag (LNCS 4948), pages 155–175, 2008.

[29] Y. Ishai, M. Prabhakaran and Amit Sahai. Secure Arithmetic Computation with No Honest Majority. In *6th TCC*, Springer-Verlag (LNCS 5444), pagess 294–314, 2009.

[30] S. Jarecki, and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *6th TCC*, Springer-Verlag (LNCS 5444), pages 577–594, 2009.

[31] S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In *EUROCRYPT'07*, Springer-Verlag (LNCS 4515), pages 97–114, 2007.

[32] E. Kiltz, P. Mohassel, E. Weinreb, and M. K. Franklin. Secure Linear Algebra Using Linearly Recurrent Sequences. In *5th TCC*, Springer-Verlag (LNCS 4392), pages 291-310, 2007.

[33] L. Kissner and D.X. Song. Privacy-Preserving Set Operations. In *CRYPTO'05*, Springer-Verlag (LNCS 3621), pages 241–257, 2005. See technical report CMU-CS-05-113 for the full version.

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

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

[36] P. Mohassel, E. Weinreb. Efficient Secure Linear Algebra in the Presence of Covert or Computationally Unbounded Adversaries. In *CRYPTO'08*, Springer-Verlag (LNCS 5157), pages 481–496, 2009.

[37] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *33th STOC* pages 590–599, 2001.

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

[39] K. Nissim, E. Weinreb. Communication Efficient Secure Linear Algebra. In *4th TCC*, pages 522–541, 2006.

[40] T. Okamoto. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In *CRYPTO'92*, Springer-Verlag (LNCS 740) pages 31–53, 1992.

[41] P. Paillier Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, Springer-Verlag (LNCS 1592), pages 223–238, 1999.

[42] T. P. Pedersen. Non-Interactive and Information-Theoretical Secure Verifiable Secret Sharing. In *CRYPTO'91*, Springer-Verlag (LNCS 576) pages 129–140, 1991.

[43] C. Peikert, V. Vaikuntanathan and B. Waters. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO'08*, Springer-Verlag (LNCS 5157), pages 554–571, 2008.

[44] C.P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 239–252, 1989.

[45] B. Vocking. How Asymmetry Helps Load Balancing. In *Journal of the ACM*, Vol. 50, No. 4, pages 568-589, 2003.

[46] A. C. Yao. Protocols for secure computations. In *23th FOCS*, pages 160-164, 1982.