Secure Function Collection with Sublinear Storage

Maged H. Ibrahim[1], Aggelos Kiayias[2], Moti Yung[3], and Hong-Sheng Zhou[2]

[1] Faculty of Engineering, Helwan University, Cairo, Egypt.
[2] University of Connecticut, Computer Science & Engineering**, Storrs, CT, USA.
E-mail: {aggelos,hszhou}@cse.uconn.edu.
[3] Google Inc. and Computer Science, Columbia University, New York, NY, USA.
Email: moti@cs.columbia.edu

Abstract. Consider a center possessing a trusted (tamper proof) device that wishes to securely compute a public function over private inputs that are contributed by some network nodes. In network scenarios that support direct communication of nodes with the center, the computation can be done by the nodes encrypting their inputs under the device's public key and sending the ciphertexts to the center which, in turn, feeds them to the trusted device that computes the function. In many modern networking scenarios, however, the center and the contributing nodes are not directly connected/ connectable, in which case the discovery and collection of inputs can only be performed by an agent (or agents) released to the network by the center. This introduces a new set of issues for secure computation. In this work we consider an agent that is released, sweeps the network once and then returns to its origin. The direct solution, in this case, is for the agent to possess a certified public key of the trusted device and for the nodes to contribute their inputs as ciphertexts under this key; once the agent collects all inputs it reconnects with the center for function computation. The above single-sweep simple collection requires the agent to store a linear number of ciphertexts. The goal of this work is to formalize and solve the above problem for a general set of functions by an agent that employs sub-linear storage while maintaining input privacy (an important technical requirement akin of that of "Private Information Retrieval" protocols).

1 Introduction

Secure multi-party computations is a general paradigm introduced by Goldreich, Micali and Wigderson [10]. It models a group of network nodes wishing to compute a public function on private inputs (or a private function on a universal circuit), producing the correct result, while a faulty subset of the nodes cannot learn from the computation more than what can be deduced from the output availability. In a model allowing nodes to also deviate from the protocol, the computation also requires that the faulty subset cannot hurt the computation.

** Research partly supported by NSF CAREER Award CNS-0447808.

This paradigm is central in cryptography and secure network protocols, and has produced numerous variations. For example, recently, it was advocated to develop solutions that take into account specific communication environments for improved efficiency [16].

In this work we consider a setting for multi-party computations which is motivated by two modern phenomena: The first is modern network environments and network processing settings where network nodes are not always connected or directly reachable by a global routing infrastructure (ad hoc networks, sensor networks), or where applications at a network node are activated only when being approached by an agent (e.g., web crawling). The second motivation is the availability of trusted computing platforms representing a trusted tamper resistant computational device. Indeed, building cryptographic systems based on trusted and tamper proof implementations and cryptographically fortifying such implementations is an area of increasing recent interest [15, 14, 9].

The problem we consider is the setting where a server (called the "center") M in possession of a tamper-proof cryptographic device wishes to compute a function described by a program P on a set of m inputs residing in network units $\mathcal{U} = \{U_1, ..., U_m\}$, to which the center has no direct interaction with. The task of collecting the inputs is given to an agent A that visits the units. The computation is required to maintain input security and to have certain robustness properties as well. The setting, which can be called "agent-oriented secure function evaluation" gives rise to new problems in the area of secure computing based on various constraints and limitations that can be put on the parties by varying the problem's parameters. In this work we consider the case of an agent that is limited to returning to the center once after sweeping the network nodes, in which case the interesting challenge is to employ an agent with sub-linear number of ciphertexts.

We formalize the above problem which we call "secure collection of a function" (SCF). We want to achieve the following security and integrity properties:

- *Unit Privacy against Malicious Agent:* Even if the agent is malicious it cannot violate the privacy of the units; furthermore,
- *Unit Privacy against Curious Center:* while the center is trusted in our framework (since it is the interested party) we still wish to maintain the privacy of the units against it (exploiting the trusted device it holds). We note that this can be particularly relevant in a setting where the center is simulated by a quorum of entities (employing threshold cryptography techniques shared among trusted units).
- *Integrity in the presence of Malicious Units:* If any number of units turn malicious they cannot collude to disrupt or spoil the computation without being detected.
- *Privacy in the presence of Malicious Units:* If any number of units turn malicious they cannot collude and learn the function that is being computed (beyond what can be deduced from their own contribution).

We do not consider the case of nodes acting against the agent. From a security point of view, the agent is oblivious to the contributions of the units and the state of the computation. Its content only reveal some encoding of state. From a robustness point of view, we note that changing the state of the agent can be detected by keeping local copies at neighbors (such extensions are left for future work). Further, in the worse case, this amounts to a total denial of service (namely, elimination of the agent), but it will not violate the privacy of the units. Mitigating the denial of service resulting by agent capturing can only be coped with by releasing a multitude of agents and we do not address these aspects at present (such variations are also suggested future work).

Our contributions and results. Here we present the concrete results we achieve. As mentioned earlier, a sweeping agent can collect all inputs in ciphertext form, a solution that requires it to carry a linear number of ciphertexts. Our goal then is to reduce this storage requirement for a large set of functions. Given our setting, we choose to implement the branching program model for representing and computing our function f. We show that this model can lead to working solutions within the constraints on agent's storage as well as those on the computational complexity imposed by the SCF setting. We employ a Paillier type encryption [24] assuming the decisional composite residuosity (DCR) assumption.

To solve our problem, we develop a new cryptographic primitive that efficiently computes a function f in the branching program model. We call it "1-out-of-2 private table evaluation" (PTE_2^1). In this primitive, the agent A has input two tables ($\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$). It delivers an encrypted version of the two tables to the unit. The unit, in turn, makes a selection according to her private input bit b, and delivers a re-encryption of her choice back to A.

Private table evaluation enables the agent to walk along a branching program layer by layer in an oblivious fashion: without ever realizing the actual state of the computation. At the end of the protocol (after visiting all units) the agent returns an encryption of the leaf representing the output of the program to the center who can then employ the trusted device to decrypt and obtain the result. During the protocol, the agent stores only an encryption of the current table. A nice property of this primitive is that the units need not have any public keys and they employ the key of the trusted device. We give an efficient implementation of the PTE_2^1 that yields an SCF protocol achieving a complexity (both in terms of communication as well as in the size of the active read/write storage of the agent) in the order of $\mathcal{O}(k \cdot w_{max})$ where k is the security parameter of the underlying homomorphic public key encryption scheme and w_{max} is the maximum width of the branching program. For a large set of branching programs this value is independent of the number of units m. For example for NC^1 programs we can get memory linear in the security parameter k due to Barrington's theorem [1]. Another appealing property of the PTE_2^1 is that, with the help of well known and suitable proofs of knowledge and with agent signing commitments to input upon its first visit we can design efficient solutions against malicious behavior of the units without hurting the complexity of the honest-but-curious protocol. The total run-time of the computation is proportional to the size of the branching program.

We show how to perform general function evaluation in this model. We then show how to implement SCF for concrete problems that are of practical importance in the setting of network sweeping: (i) polling-to-partition procedures, that can simulate a wide array of distributed decision-making and polling operations, and (ii) pattern and string matching procedures typical in search problems. Finally, we note that to further hide the circuit structure from units, the agent can perform dummy PTE_2^1 operations, in a random walk, mixing it with real computations.

Related Work. Secure multiparty computation was initiated and studied by Goldreich, Micali and Wigderson [10] in a fully connected (broadcast links). This was followed by information theoretic protocols assuming (added) private links [2, 4, 26]. Various models with partial connectivity were then considered in [7, 16, 8]. In our setting we take the approach to restrict even further the connectivity (by not assuming a complete routing infrastructure as in ad hoc scenarios). We further employ a trusted device (also employed in recent work on employing trusted hardware in [15, 11]). We next note that our problem bears some relation to private information retrieval [5, 19, 3] where a client wishes to extract a record from a database in sublinear communication. The distinction in our setting is that the role of the database is distributed to a set of units and the role of the communication channel is played by the agent that interacts with the units.

The use of the branching program model of computation for reducing the communication complexity in secure computations between two curious parties was put forth by Naor and Nissim in [22]. The model was also utilized in the context of sublinear communication oblivious transfer in [21] and for general two party computation in [13].

Organization: In section 2 we the formulation of the problem and the security model. Next, in section 3 we show our basic protocol secure collection of a function protocol construction using private table evaluation as a building block. An explicit private table evaluation protocol is then presented in section 4. Due to lack of space, examples of our framework are presented in the appendix in section A. The zero-knowledge proof protocol for enforcing security against malicious units is presented in section B and proofs of our claims are presented in section C.

2 Problem and Model

We first introduce the general formulation of our problem.

Definition 1. *The Collecting a Function (CF) problem: A* center M *wishes to compute a function $f : X^m \rightarrow Z$ on inputs $x_1, \ldots, x_m$ that are distributed to a set of units $\mathcal{U} = \{U_1, ..., U_m\}$. The units communicate with the center indirectly through black-box interaction with an agent A.*

A communication pattern Π for a CF problem is a string P from $\{U_1, \ldots, U_m\}^*$ that specifies the order with which the agent interacts with the units. Note that the agent must also interact two times with the center, one time when the agent *is dispatched to collect the information (initialization) and another time when the agent delivers the results to the center.*

A secure CF (SCF) protocol for a function $f : X^m \rightarrow Z$ and a communication pattern P is a triple $\langle \mathsf{Init}, \mathsf{Final}, (\pi_A, \pi_U)\rangle$ such that Init *receives a description of f and a security parameter and produces a public/secret-key pair (pk, sk); (π_A, π_U) is a two-party protocol where in the j-th round, ($j \in \{1, \ldots, |P|\}$), the agent running π_A on input (s_{j-1}, pk) and a description of f interacts with the unit U_t, where $t \in \{1, \ldots, m\}$, running protocol π_U on input (s'_{j-1}, x_t); the two parties terminate returning private outputs s_j and s'_j respectively. Note that $s_0 := \epsilon$, and $s'_0 := pk$.* Final *receives the string $s_{|P|}$ and sk and returns a value in the range of f.*

An SCF protocol is correct if for all $x_1, \ldots, x_m \in X$ it holds that Final *in the computation as described above returns the value $f(x_1, \ldots, x_m)$ always.*

Security Properties for SCF protocols. Next we introduce the security and integrity properties for SCF protocols.

- **Unit Privacy against a Malicious Agent.** For any $i = 1, \ldots, m$, and any $x_1, \ldots, x_m \in X$, and any PPT adversary Adv playing the role of the agent A and all units $\{U_1, \ldots, U_m\} \setminus \{U_i\}$, there exists an expected polynomial-time simulator Sim, such the output of Sim is computationally indistinguishable to the view of the adversary Adv that includes all protocol transcripts and private tapes of corrupted parties. The input of Sim is equal to the input of the (corrupted) agent A and the (corrupted) units $\{x_1, \ldots, x_m\} \setminus \{x_i\}$.
- **Unit Privacy against a Honest-but-Curious Center.** For any $x_1, \ldots, x_m \in X$, there exists an expected polynomial-time simulator Sim such that the output of Sim is identical to the distribution of the internal state of the agent at the end of all interactions with the units. The input of Sim is equal to the value $f(x_1, \ldots, x_m)$.
- **Computation Integrity in the presence of Malicious Units.** There is a deterministic implicit input function IN, such that for any PPT adversary Adv, playing the role of all units $\{U_1, \ldots, U_m\}$, it holds that if the agent terminates successfully then the output of the center M equals $f(x_1, \ldots, x_m)$ where $(x_1, \ldots, x_m) = \mathsf{IN}(\tau)$ where τ is the total communication transcript of all the interactions between the agent and the units. Note that IN is not required to be polynomial-time.
- **Computation Privacy in the presence of Malicious Units.** For any $i = 1, \ldots, m$, and any $x_1, \ldots, x_m \in X$, and any PPT adversary Adv playing the role of the unit U_i, there exists an expected polynomial-time simulator Sim, such the output of Sim is computationally indistinguishable to the view of the adversary Adv that includes all protocol transcripts between the unit U_i and the agent A.

Efficiency Parameters of SCF protocols. For an SCF protocol $\langle \mathsf{Init}, \mathsf{Final}, (\pi_A, \pi_U)\rangle$ we are interested in the following efficiency measures:

- Agent Storage. Is the maximum number of storage bits required by the agent quantified over all possible CF computations and coin tosses.
- Agent-Unit Communication. Is the maximum number of bits communicated between the unit and the agent quantified over all possible SCF protocol computations and coin tosses.

The agent is also limited in computation and resources. The problem also considers privacy against the agent and assumes the agent is honest-but-curious in the sense that he follows the execution steps of the protocol word for word but he is willing to learn whatever could be revealed during execution. The units are assumed honest about their own inputs but they could contribute maliciously in order to drift the computation from the correct progress (e.g. the program output becomes invalid).

3 Designing SCF protocols from Branching Programs

We first describe the notion of layered branching programs for modelling multi-party protocols. The definition below is a straightforward generalization of the notion of two party branching programs used for modelling protocols in the communication complexity model as given in [18]. We define how m parties can compute a function $f : X^m \to Z$ based on their inputs and broadcast communication.

Definition 2 (Multiparty computation in the Branching Program model (BPM)). *A (layered) branching program P for the function $f : X^m \to Z$ and parties $U_1, \ldots, U_m$ is a layered directed graph $G = (V = (L_0, ..., L_c), E)$ of depth c where $|L_0| = 1$. Each edge in E connects layer L_ℓ to layer $L_{\ell+1}$ where $\ell \in \{0, \ldots, c-1\}$ and is labeled by an element of X so that each vertex has exactly $|X|$ outgoing edges all labeled distinctly. Further, each layer is labeled by a party $U \in \{U_1, \ldots, U_m\}$ and each node in L_c is labeled with an element $z \in Z$.*

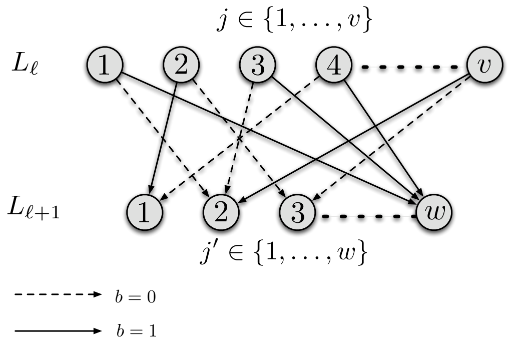
The output of the program P on input $(x_1, ..., x_m)$ where $x_i \in X$ is the label of the leaf reached by starting at the single node at the L_0 level and traversing the graph following the appropriately labeled edges at each level based on the input of the party that labels the layer. We say that the branching program P computes $f : X^m \to Z$ if its value on input $(x_1, ..., x_m)$ equals $f(x_1, ..., x_m)$. The cost of the program is c, its width is $\max |L_\ell|$.

A branching program P can capture multiparty computation of a function f in the following way: the party U_i that labels the top layer L_0 that contains a single node v_0 computes the next node following the edge labeled according to its input and broadcasts the index of the next node to the other parties. Next, the party at level L_1 continues recursively based on the communication transcript and the computation proceeds recursively until all parties terminate at the leaf level. The overall communication complexity equals c broadcasts of at most $\log_2 \max\{|L_\ell|\}$ bits.

In our setting the units do not interact with one another but rather the broadcast channel can be simulated by interacting with the agent. In particular:

- Given the branching program P of cost c for the function $f : X^m \to Z$, we derive a communication pattern $\Pi \in \{U_1, \ldots, U_m\}^c$ based on the labels of the layers of P.
- The agent maintains a state that corresponds to a node of the branching program P. Initially this node is set to be single node v_0 of the top layer L_0.
- The agent, while at a state corresponding to the v-th vertex at level L_ℓ for $\ell \in \{0, \ldots, c-1\}$, visits the next unit U_i following the order of the communication pattern Π and executes a protocol with U_i that enables the agent to compute the index of the vertex in level $L_{\ell+1}$ in the layered branching program P that is determined by the outgoing edge of v labeled by x_i where x_i is the input of the unit U_i.
- The computation proceeds recursively until the agent reaches the last layer of the branching program P that reveals the value $f(x_1, \ldots, x_m)$.

In the remaining of the section for simplicity we will focus on the special case that $X = \{0, 1\}$. Extending our results to the general case is straightforward. We next consider how one can add privacy to the above CF computation strategy. We first observe that each level of the branching program can be represented by a set of tables $\mathcal{T}^{(0)}, \mathcal{T}^{(1)}$. We depict this in figure 1.



j	j'
1	2
2	3
3	2
4	1
⋮	⋮
v	3

j	j'
1	w
2	1
3	w
4	w
⋮	⋮
v	2

Fig. 1. An arbitrary level in a fragment of branching program (on the left) and the corresponding tables (on the right). The choice of a table depends on U's input b.

At each layer of the computation as described above, the agent and the unit will utilize a protocol that we call *1-out-of-2 private table evaluation* (PTE). This is a two-party protocol that relies on a suitable public-key encryption scheme $\langle \mathcal{G}, \mathcal{E}, \mathcal{D} \rangle$. The agent will be given its input j in encrypted form as $\mathcal{E}_{pk}(j)$ and in the course of the interaction with the unit it expects to update it to the encryption $\mathcal{E}_{pk}(\mathcal{T}^{(b)}[j])$.

The two tables $\mathcal{T}^{(0)}, \mathcal{T}^{(1)}$ are selected from $\{1, \ldots, w\}^v$ where $v, w \in \mathbb{N}$ are parameters of the protocol. For greater generality in our protocol description we will also allow the underlying encryption scheme used to be parameterized and use the notation $\mathcal{E}^v_{pk}(m)$ for any $v \in \mathbb{N}$. Pictorially we have the following:

Definition 3 (1-out-of-2 private table evaluation (PTE^1_2)). *The protocol is based on parameters $v, w \in \mathbb{N}$ and is executed by agent A and unit U based*

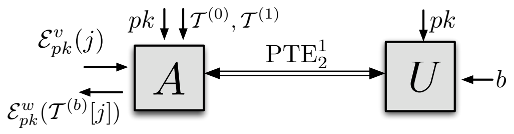

pk
$\mathcal{T}^{(0)}, \mathcal{T}^{(1)}$
$\mathcal{E}^v_{pk}(j)$
$\mathcal{E}^w_{pk}(\mathcal{T}^{(b)}[j])$
A
PTE^1_2
U
pk
b

Fig. 2. 1-out-of-2 private table evaluation.

on a public key encryption $\langle \mathcal{G}, \mathcal{E}, \mathcal{D} \rangle$. First an external party generates a key pair $(pk, sk) \leftarrow \mathcal{G}(1^\lambda)$, and delivers pk to both agent A and unit U. Second A interacts with U; here A has input $\mathcal{E}^v_{pk}(j)$ where $j \in \{1, \ldots, v\}$, and $\mathcal{T}^{(0)}, \mathcal{T}^{(1)} \in \{1, \ldots, w\}^v$, and U has the input $b \in X$. Finally only the agent outputs $\mathcal{E}^w_{pk}(j')$ where $j' = \mathcal{T}^{(b)}[j] \in \{1, \ldots, w\}$.

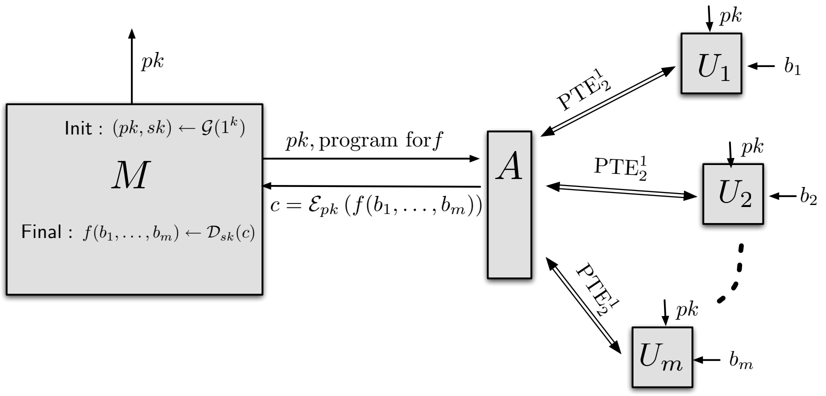
We will say that a PTE^1_2 protocol is secure, if the following properties are satisfied. In all cases below it is assumed that the input of both parties includes the public-key pk that is honestly generated and neither party is in possession of the secret-key.

- **Integrity.** Informally, we require that the unit can only contribute in terms of selecting one of the two tables that the agent possesses and is not capable – even if acting maliciously – to influence the agent to derive an encryption of any value other than an encryption of j'. More formally, we require that even if the unit is malicious it holds that the output of the agent is distributed uniformly either over all ciphertexts of the form $\mathcal{E}_{pk}(\mathcal{T}^{(0)}[j])$ or all ciphertexts of the form $\mathcal{E}_{pk}(\mathcal{T}^{(1)}[j])$.
- **Unit Privacy.** Informally, we require that the input of the unit remains hidden from the agent. More formally, for any $x \in X$, and any PPT adversary Adv corrupting the agent A, there exists an expected polynomial-time simulator Sim, such that the output of Sim is computationally indistinguishable to the view of the adversary Adv that includes protocol transcripts and private tapes of the corrupted agent. The input of Sim equals the corrupted agent's input.
- **Agent Privacy.** Informally, we require that the two input tables $\mathcal{T}^{(0)}, \mathcal{T}^{(1)}$ of the agent remain hidden from the unit. More formally, for any $x \in X$, and any PPT adversary Adv corrupting the unit U, there exists an expected polynomial-time simulator Sim, such the output of Sim is computationally indistinguishable to the view of the adversary Adv that includes protocol transcripts between the unit U and the agent A.

Our Design Strategy. Based on an implementation of PTE^1_2 we can derive a SCF protocol for any function $f : X^m \to Z$ as follows. Let P be a branching program for computing f with communication pattern Π. We assume each unit U_i holds an input $b_i \in X = \{0, 1\}$. Our SCF system is illustrated in figure 3 and operates as follows:

- In the initialization process Init, given a security parameter k a key pair $(pk, sk) \leftarrow \mathcal{G}(1^k)$. The public key pk is supplied to the agent A and all units, and the secret key sk is only known by the center. M also gives A the branching program P for function f.

- According to the structure of P, A invokes the PTE_2^1 with each unit $U \in \{U_1, \ldots, U_m\}$. In particular at each layer of P (except the last) the agent defines two tables $\mathcal{T}^{(b)}[\cdot] \in \{1, \ldots, w\}^v$ where v is the width of the current layer and w is the width of the next layer as shown in figure 1.
- Using sk, Final decrypts the ciphertext to receive the decryption of the final index that also reveals the output of the computation.



pk
Init : $(pk, sk) \leftarrow \mathcal{G}(1^k)$
M
Final : $f(b_1, \ldots, b_m) \leftarrow \mathcal{D}_{sk}(c)$
pk, program for f
$c = \mathcal{E}_{pk}(f(b_1, \ldots, b_m))$
A
PTE_2^1
PTE_2^1
PTE_2^1
pk
U_1
b_1
pk
U_2
b_2
pk
U_m
b_m

Fig. 3. The SCF protocol based on private-table evaluation.

Theorem 1. *The SCF protocol defined above satisfies (i) unit privacy against a malicious agent under the assumption that the given PTE satisfies unit privacy (ii) unit privacy against an honest-but-curious center unconditionally (iii) computation integrity in the presence of malicious units under the assumption that the given PTE satisfies integrity, (iv) computation privacy in the presence of malicious units under the assumption that the given PTE satisfies agent privacy.*

4 Implementing Private Table Evaluation Efficiently

Our implementation of PTE_2^1 is based on an IND-CPA public-key encryption scheme with homomorphic property. Here we use the Paillier cryptosystem [24]; we note that the construction could be based on other homomorphic encryptions. Next we give a brief review of Paillier system $\langle \mathcal{G}, \mathcal{E}, \mathcal{D} \rangle$.

- Key Generation, $\mathcal{G}(1^k)$. On input a security parameter k, let p, q be random k-bit primes such that $p, q > 2$, $p \neq q$, and $\gcd(pq, (p-1)(q-1)) = 1$; let $N = pq$, $d = \mathrm{lcm}(p-1, q-1)$, $K = d^{-1} \bmod N$, and $h = (1+N)$; output a key pair (pk, sk), where $sk = \langle p, q \rangle$ and $pk = \langle N, h \rangle$.
- Encryption, $\mathcal{E}_{pk}(m)$. Define a plaintext space $\mathcal{M}_{pk} := \mathbb{Z}_N$ and a ciphertext space $\mathcal{C}_{pk} := \mathbb{Z}_{N^2}^*$. Upon receiving $m \in \mathcal{M}_{pk}$, randomly choose $r \in_R \mathbb{Z}_N^*$, and output the ciphertext $c = h^m r^N \bmod N^2$. We denote the ciphertext as $\mathcal{E}_{pk}(m)$ or $\mathcal{E}_{pk}(m; r)$.
- Decryption, $\mathcal{D}_{sk}(c)$. Upon receiving $c \in \mathcal{C}_{pk}$, compute $m = \frac{(c^{dK} \bmod N^2) - 1}{N}$ $\bmod N$. Note that d, K can be obtained from sk as in key generation.

The Paillier cryptosystem is IND-CPA secure [24] under the Decisional Computational Residuosity (DCR) assumption. It also enjoys an additive homomorphic property. For binary operators "+" over $\mathcal{M}_{pk}$ and "$\cdot$" over $\mathcal{C}_{pk}$, we have $\mathcal{E}_{pk}(m_1; r_1) \cdot \mathcal{E}_{pk}(m_2; r_2) = \mathcal{E}_{pk}(m_1 + m_2; r_1 r_2)$.

Observe that based on the homomorphic property, we can randomize a valid ciphertext c into another one c' while preserving the underlying plaintext m intact. In particular, c' can be obtained by multiplying c with a ciphertext for 0, i.e., $c' = c \cdot \mathcal{E}_{pk}(0; r_0) = \mathcal{E}_{pk}(m; r_m) \cdot \mathcal{E}_{pk}(0; r_0) = \mathcal{E}_{pk}(m; r_m r_0)$, where $r_m, r_0 \in_R \mathbb{Z}_N^*$. Notice that for any c it holds that c' is a random variable over the subset of $\mathcal{C}_{pk}$ that is of size $|\mathbb{Z}_N^*|$ and corresponds to all possible encryptions of the plaintext in c.

The parameterization we will use will be as follows: $\mathcal{E}^v_{pk}(m) = \langle \mathcal{E}_{pk}(m), \mathcal{E}_{pk}(m^2), \ldots, \mathcal{E}_{pk}(m^{v-1}) \rangle$, i.e., for any $v > 0$ it holds that $\mathcal{E}^v_{pk}(m)$ is a vector of $v - 1$ ciphertexts. Note that if $v = 1$ it holds that $\mathcal{E}^1_{pk}(m) = \langle \mathcal{E}(1) \rangle$ (independently of m).

4.1 Building block: a PTE construction

We present next our construction for private table evaluation that is utilizing Paillier encryption as defined in the previous section.

1. Generate $(pk, sk) \leftarrow \mathcal{G}(1^k)$, deliver pk to agent A and unit U. Then A and U operate as follows.
2. Upon receiving two tables $\mathcal{T}^{(0)}, \mathcal{T}^{(1)} \in \{1, \ldots, w\}^v$, and a vector of ciphertexts $\mathcal{E}^v_{pk}(j) = \langle C_1, \ldots, C_{v-1} \rangle$ where recall that $C_i = \mathcal{E}_{pk}(j^i)$ for $i = 1, \ldots, v - 1$, $j \in \{1, \ldots, v\}$, the agent A chooses $2(w - 1)$ polynomials with degree $v - 1$ as: $g^{(0)}_1(x), \ldots, g^{(0)}_{w-1}(x), g^{(1)}_1(x), \ldots, g^{(1)}_{w-1}(x)$ over $\mathbb{Z}_N^*$, such that $g^{(b)}_\ell(j) = (\mathcal{T}^{(b)}[j])^\ell \bmod N$, for $j = 1, \ldots, v$, where $\ell \in \{1, \ldots, w - 1\}$ and $b \in \{0, 1\}$. Notice that the polynomial $g^{(b)}_\ell(x)$ is computed as $g^{(b)}_\ell(x) = \sum_n \lambda^{(j)}_n (\mathcal{T}^{(b)}(n))^\ell$ where the $\lambda^{(j)}_n$'s are the Lagrange coefficients: for any polynomial p of degree less than v they satisfy $p(x) = \sum_{n=0}^{v-1} \lambda^{(x)}_n p(n)$. Let $a^{(b,\ell)}_0, \ldots, a^{(b,\ell)}_{v-1}$ be the coefficients of polynomial $g^{(b)}_\ell(x)$.
   Using the above set of polynomials and its input ciphertexts $\langle C_1, \ldots, C_{v-1} \rangle$ the agent computes $C^{(b)}_\ell = \mathcal{E}_{pk}(g^{(b)}_\ell(j)) = \prod_{i=0}^{v-1} (C_i)^{a^{(b,\ell)}_i}$, where $C_0 = \mathcal{E}_{pk}(1)$ for $\ell = 1, \ldots, w - 1$. We denote $C^{(b)} = \langle C^{(b)}_1, \ldots, C^{(b)}_{w-1} \rangle$, for $b = 0, 1$. Observe that $C^{(b)} = \mathcal{E}^w_{pk}(\mathcal{T}^{(b)}[j])$. The agent sends $\langle C^{(0)}, C^{(1)} \rangle$ to unit U.
3. On input $b \in \{0, 1\}$, after receiving $\langle C^{(0)}, C^{(1)} \rangle$ from A, the unit randomizes $C^{(b)}$ into C' where $C' = \langle C'_1, \ldots, C'_{w-1} \rangle$ and $C'_\ell = C^{(b)}_\ell \cdot \mathcal{E}_{pk}(0; r_\ell)$ and $r_\ell \in_R \mathbb{Z}_N^*$ for $\ell = 1, \ldots, w - 1$, and returns C' to agent A.
4. Finally agent A outputs C'.

Below we prove the security of the above protocol for honest units. In the next session we show how to *efficiently* handle malicious units. We show:

Theorem 2. *The PTE protocol defined above satisfies (i) integrity for honest units unconditionally, (ii) unit privacy under DCR assumption, (iii) agent privacy under DCR assumption.*

4.2 PTE against malicious units

To defend against a malicious unit, we need to enforce the unit to commit to its input b, i.e, $E = h^b t^N \bmod N^2$ where $t \in_R \mathbb{Z}_N^*$, and the commitment E is further to be signed by the agent, $\sigma = \texttt{Sign}_{sk}(E)$. Further the unit needs to prove that the vector of ciphertexts C' is randomization of $C^{(b)}$ and such b is the one committed in E, that means the unit should show a proof that it knows the witness of the following language

$$\mathcal{L} = \left\{ \begin{array}{l} \langle C_1^{(0)}, \ldots, C_{w-1}^{(0)}, C_1^{(1)}, \ldots, C_{w-1}^{(1)}, C_1', \ldots, C_{w-1}', E \rangle | \\ \qquad \exists b, r_1, \ldots, r_{w-1}, \beta : \\ \qquad\quad C_\ell'/C_\ell^{(0)} = (C_\ell^{(1)}/C_\ell^{(0)})^b \cdot r_\ell^N \bmod N^2 \text{ for } \ell = 1, \ldots, w-1, \\ \qquad\quad E = h^b \beta^N \bmod N^2 \end{array} \right\}$$

Note that the unit should save $\langle E, \sigma \rangle$ and in the case that the agent visits the same unit again, $\langle E, \sigma \rangle$ will be handed to the agent and the ZK proof will be based on this recorded E and newly produced vector of ciphertexts. The proof can be done by standard techniques, please refer to the appendix for more details.

Theorem 3. *The PTE protocol defined above satisfies (i) integrity under the* `Sign` *unforgeability, (ii) unit privacy under DCR assumption, (iii) agent privacy under DCR assumption.*

Corollary 1. *The SCF protocol of section 3 with the PTE of the present section satisfies (i) unit privacy against a malicious agent under DCR assumption (ii) unit privacy against a semi-honest center unconditionally (iii) computation integrity in the presence of malicious units under* `Sign` *unforgeability, (iv) computation privacy in the presence of malicious units under DCR assumption.*

References

1. D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.
2. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.
3. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414, 1999.
4. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.
5. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

6. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, pages 103–118, 1997.
7. D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, 1993.
8. J. A. Garay and R. Ostrovsky. Almost-everywhere secure computation. In *EUROCRYPT*, pages 307–323, 2008.
9. R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.
10. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
11. C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standardsmartcards. In *ACM CCS*, pages 491–500, 2008.
12. A. Hevia and M. A. Kiwi. Electronic jury voting protocols. In *LATIN*, pages 415–429, 2002.
13. Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.
14. Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private circuits II: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.
15. J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.
16. J. Katz and C.-Y. Koo. Round-efficient secure computation in point-to-point networks. In *EUROCRYPT*, pages 311–328, 2007.
17. A. Kiayias and M. Yung. Non-interactive zero-sharing with applications to private distributed decision making. In *Financial Cryptography*, pages 303–320, 2003.
18. E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.
19. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.
20. M. Langberg, A. Pnueli, and Y. Rodeh. The ROBDD size of simple CNF formulas. In D. Geist and E. Tronci, editors, *CHARME*, volume 2860 of *LNCS*, pages 363–377. Springer, 2003.
21. H. Lipmaa. An oblivious transfer protocol with log-squared communication. In *ISC*, pages 314–328, 2005.
22. M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *STOC*, pages 590–599, 2001.
23. J. B. Nielsen. On protocol security in the cryptographic model. In *Dissertation Series DS-03-8, BRICS*, 2003.
24. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.
25. P. W. C. Prasad, A. K. Singh, and A. Assi. ROBDD complexity analysis for XOR/XNOR min-terms. In *International Journal of Electronics*, volume 95, pages 111–123, 2008.
26. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85. ACM, 1989.

A Examples

In this section we introduce several useful instances of securely collecting a function.

A.1 General decision-making procedures

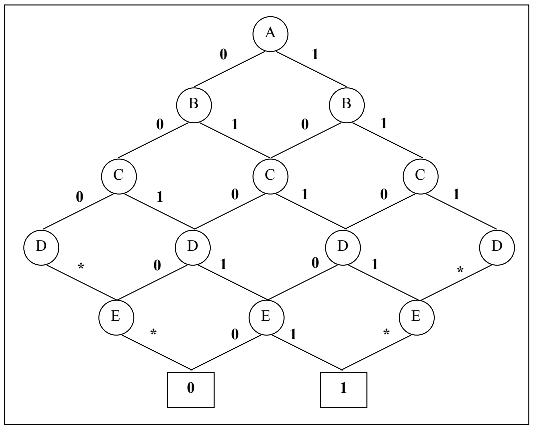


Fig. 4. Example of a branching program for computing the majority function; in the example, there are five units; each unit holds one private bit. The center constructs a five level branching program and loads the agent with the program. Then the agent incorporates with each unit in a PTE_2^1. After visiting all units, the agent returns the result which is the encryption of either leaf (0) or leaf (1). The center then decrypts what he receives from the Agent. If the result of the decryption is 0 then the majority of the bits held by the units is 0, otherwise, the majority is 1. The symbol (*) indicates a "don't care" condition i.e. this branch will be followed whether the corresponding input is either 0 or 1.

Suppose that each unit $U_i \in \mathcal{U}$ of the m units holds a private bit $b_i \in \{0, 1\}$, with $i = 1, \ldots, m$. A "polling-to-partition" procedure is defined over a partition $A_1, \ldots, A_k$ of the set $\{0, \ldots, m\}$ and requires the calculation of the index j for which it holds that $\sum_{i=1}^{m} b_i \in A_j$.

Some examples of decision-making procedures that can be modeled as polling-to-partition procedures involve the interpretation of the choice of each unit as a yes/no answer to a polling question and are as follows: (i) polling-to-partition over $\{\{0\}, \ldots, \{m\}\}$ will calculate the exact number of "yes" choices by the units, (ii) polling-to-partition over $\{\{0, \ldots, \lfloor m/2 \rfloor\}, \{\lfloor m/2 \rfloor + 1, \ldots, m\}\}$ will provide a way to test whether the majority of units agree or disagree; (iii) polling-to-partition over $\{\{0, 1, \ldots, m-1\}, \{m\}\}$ provides a way to test whether there is consensus between the units.

We observe that for each partition it is possible to derive an unordered branching program that is of length m and has exactly k nodes in the final level (equal to the number of elements in the partition). The width of the branching program varies but in any case it is $\mathcal{O}(m - \max_j\{|A_j|\})$. In figure 4, we show the branching program for the majority computation. The example shows the branching program sampled by the center for five units. There are only two leafs in this program. Leaf (0) is reached if the majority of unit inputs is 0 while leaf (1) is reached if the majority of unit inputs is 1. In a similar way we can construct the branching programs of all other polling-to-partition procedures.

Using our SCF protocol it is possible for the center to calculate the output of any polling-to-partition procedure without revealing any information about the choices $b_1, \ldots, b_m$ of the units beyond the index of the partition set that the sum $\sum_{i=1}^{m} b_i$ belongs to. We note that simulating specific instances of polling-to-partition procedures were feasible before using other techniques: in particular, one can perform yes/no voting directly with homomorphic encryption, see e.g., [6], or calculate whether overwhelming majority exists, see e.g., [12], or test consensus [17]. Nevertheless such solutions either don't apply to important instances of polling-to-partition procedures such as testing whether a majority exists or they rely on network configurations such as mix-networks that are not consistent with our application domain.

A.2 String matching

In the case of string matching, a string $\alpha \in \{0,1\}^c$ is distributed among the units so that each unit U_i holds a substring α_i where $\alpha = \alpha_1||\alpha_2...||\alpha_m$. The center wants to know whether α is accepted by the program $\mathcal{A}$, where $\mathcal{A}$ is a private deterministic finite automaton $\mathcal{A}= (Q, \delta)$ where $Q = \{S_0, ..., S_n\}$ and $\delta : Q \times \{0,1\} \rightarrow Q$. It is possible to define a branching program based on $\mathcal{A}$ with cost $\mathcal{O}(c)$ and width $\mathcal{O}(n)$. The program considers a single bit of α at a time and hence each unit U_i is assigned one or more consequent levels on the program according to $|\alpha_i|$. The program stores the state (either accepting or rejecting) reached by all prefixes of α. The root node L_0 is labeled S_0 and each intermediate level L_i is labeled by Q for $1 \leq i \leq c$. The terminal nodes (leafs) in L_n are just two and correspond to the accepting and rejecting states. The agent returns the result to the center who decrypts for the leaf and decides whether or not the string α is accepted by $\mathcal{A}$. In figure 5, we show a simple example in which each unit holds a single bit, in this example, the string to be matched is 1100. Other finite automata can facilitate pattern matching and other string operations.

A.3 Computing an arbitrary function

Figure 6 shows a truth table for an arbitrary function f and the standard method to convert this truth table to an efficient reduced-ordered-binary-decision-diagram ROBDD (that happens to have width 2 in this case). First, the truth table is constructed for f as in figure 6(a), then, a full binary decision tree is constructed

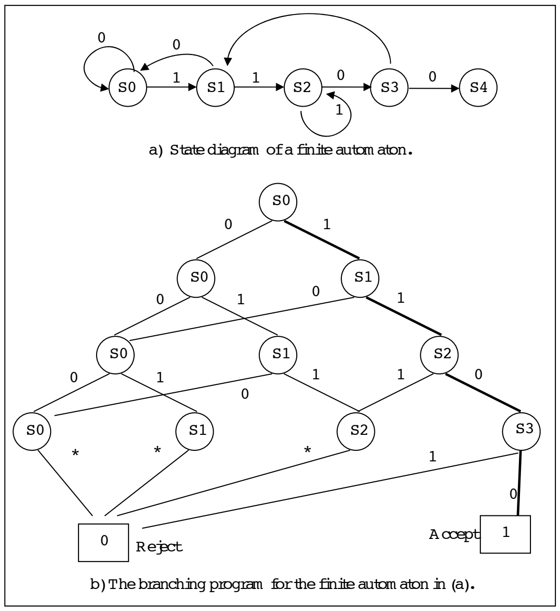
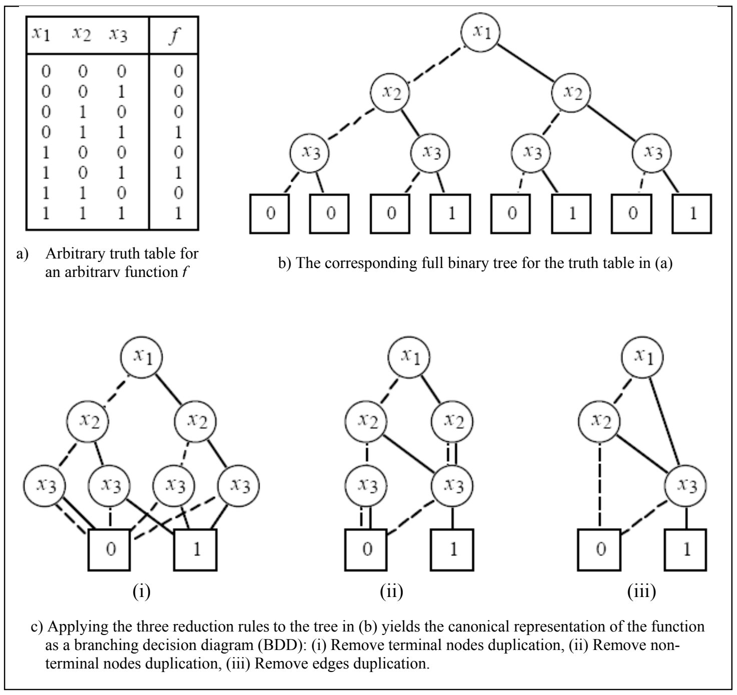

S0
S1
S2
S3
S4
0
1
a) State diagram of a finite automaton.
*
Reject
Accept
b) The branching program for the finite automaton in (a).

Fig. 5. An example of an oblivious branching program for "string matching" the string in the example is 1100; Given a specific string, the output of the program is either (accept) if the string is found or (reject) otherwise. In the example, the output is (accept).

from the truth table as in figure 6(b), finally, three reduction rules are applied to the tree to reach the final ROBDD as in figure 6(c). The problem is that, as the size of the input grows, the size of the truth table blows up. Therefore, it is better to start from a circuit (CNF or DNF) formula and converts the formula to an efficient branching programs. There are many techniques to reach an efficient ROBDD from a given circuit, see e.g., [20, 25].



x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

a) Arbitrary truth table for an arbitrary function f

b) The corresponding full binary tree for the truth table in (a)

c) Applying the three reduction rules to the tree in (b) yields the canonical representation of the function as a branching decision diagram (BDD): (i) Remove terminal nodes duplication, (ii) Remove non-terminal nodes duplication, (iii) Remove edges duplication.

Fig. 6. Example of an arbitrary function and the corresponding branching program.

B ZK proof in the PTE against malicious units

Here we give more details for the PTE construction against malicious units. Following steps 1,2 in the basic PTE construction in section 4.1.

3. On input $b \in \{0,1\}$, after receiving $\langle C^{(0)}, C^{(1)} \rangle$ from A, the unit randomizes $C^{(b)}$ into C' where $C' = \langle C'_1, \ldots, C'_{w-1} \rangle$ and $C'_\ell = C^{(b)}_\ell \cdot \mathcal{E}_{pk}(0; r_\ell)$ and $r_\ell \in_R \mathbb{Z}^*_N$ for $\ell = 1, \ldots, w-1$. If the unit is interacted by the agent for the first time, then it further computes commitment $E = h^b t^N \bmod N^2$ where $t \in_R \mathbb{Z}^*_N$, and saves $\langle E, t \rangle$ for future use, and the unit returns ciphertext vector C' and also commitment E to agent A. If the unit has been visited before, then it retrieve previous recorded E, t, σ and returns C' and E,σ to the agent; here σ is the previously obtained signature for E from the agent.
4. The unit develops an efficient (interactive) zero-knowledge proof with the agent to show that C' is the correct re-encryption of $C^{(b)}$ and based on same b as committed into E. The zero-knowledge proof details can be found blow.
5. If the unit has been visited before, then the agent besides verifies the ZK proof, and also need to verify that σ is a valid signature for E. Else if the unit is the first time to be visited, then the agent A signs the commitment value E, i.e., $\sigma = \mathtt{Sign}_{sk}(E)$, and returns σ to the unit; the agent also outputs the ciphertext vector C'.
6. Upon receiving σ, if it can be verified, then the unit records it.

We still need to describe the zero-knowledge proof mentioned in step 4. The unit should show a proof that it knows the witness of the following language

$$\mathcal{L} = \left\{ \begin{array}{l} \langle C^{(0)}_1, \ldots, C^{(0)}_{w-1}, C^{(1)}_1, \ldots, C^{(1)}_{w-1}, C'_1, \ldots, C'_{w-1}, E \rangle | \\ \qquad \exists b, r_1, \ldots, r_{w-1}, t : \\ \qquad\quad C'_\ell / C^{(0)}_\ell = (C^{(1)}_\ell / C^{(0)}_\ell)^b \cdot r^N_\ell \bmod N^2 \text{ for } \ell = 1, \ldots, w-1, \\ \qquad\quad E = h^b t^N \bmod N^2 \end{array} \right\}$$

This can be done by design an honest-verifier zero-knowledge proof and then by applying a commitment to defend against a malicious verifier. The reader may refer to e.g., Section 2.9 of [23] for a discussion of this technique.

Step 4 can be divided into four substeps. Note that for each ℓ, $C'_\ell / C^{(0)}_\ell = (C^{(1)}_\ell / C^{(0)}_\ell)^b \cdot r^N_\ell \bmod N^2$. In substep 4.1, the unit computes $\alpha_\ell = (C^{(1)}_\ell / C^{(0)}_\ell)^{\beta_\ell} \cdot \gamma^N_\ell \bmod N^2$ where $\beta_\ell \in_R \mathbb{Z}_N$ and $\gamma_\ell \in_R \mathbb{Z}^*_N$. Note that $E = h^b t^N \bmod N^2$; the unit also computes $\delta = h^\eta \tau^N \bmod N^2$ where $\eta \in_R \mathbb{Z}_N$ and $\tau \in_R \mathbb{Z}^*_N$. Further the unit commits α_ℓ's and δ into a commitment value z, i.e., $(z, \zeta) \leftarrow \mathtt{com}(\alpha_1, \ldots, \alpha_{w-1}, \delta)$ where ζ is the opening of the commitment. Now the unit sends z to the agent.

In substep 4.2, the agent randomly selects $d \in_R \{0,1\}^\kappa$ for the unit.

In substep 4.3, after receiving d from the agent, the unit computes $\xi_\ell = \beta_\ell - d \cdot b$ and $\varphi_\ell = \gamma_\ell \cdot r^{-d}_\ell \bmod N$ for each ℓ, and computes $\theta = \eta - d \cdot b$ and $\rho = \tau \cdot t^{-d} \bmod N$. The unit returns all ξ_ℓ's and φ_ℓ's to the agent, and the unit also open the commitments in step 4.1 to the agent.

In substep 4.4, the agent checks if the commitment is verified, and $\alpha_\ell = (C'_\ell/C^{(0)}_\ell)^d(C^{(1)}_\ell/C^{(0)}_\ell)^{\xi_\ell}\varphi_\ell^N \bmod N^2$ for all ℓ, and $\delta = E^d h^\theta \rho^N \bmod N^2$.

Note that the commitment scheme $\mathtt{com}()$ used in the protocol can be chosen to be unconditionally binding.

C Proofs

Proof (Proof sketch of theorem 1). Under the assumption that the given PTE protocol satisfies unit privacy, it is straightforward to show unit privacy against a malicious agent. We can construct a simulator Sim as follows. First the simulator runs a copy of adversary Adv internally which corrupts A and units $\{U_1, \ldots, U_m\} \setminus \{U_i\}$. The corrupted units can be simulated given their real inputs $\{x_1, \ldots, x_m\} \setminus \{x_i\}$ are supplied to Sim. The simulator further simulates an honest center M to send pk and two tables $\mathcal{T}^{(0)}$, $\mathcal{T}^{(1)}$, and also a vector of ciphertexts to $\mathsf{Sim}_{\mathrm{PTE}}$, which simulates the corrupted A internally; here $\mathsf{Sim}_{\mathrm{PTE}}$ is the simulator for unit privacy in PTE protocol. Now the adversary's view is indistinguishable from the simulated view.

Next we consider unit privacy against a semi-honest center. The simulator Sim runs a semi-honest adversary Adv internally which controls M. After receiving pk, f from the corrupted M, the simulator replies it with an encryption of $f(x_1, \ldots, x_m)$. Note that $f(x_1, \ldots, x_m)$ is given to Sim. Now the adversary's view is identical to the simulated view.

Given that PTE protocol satisfies integrity, each unit has contribution to the computation, then M can compute $f(x_1, \ldots, x_m)$ where x_i is unit U_i's contribution. Note that an unbounded IN machine can extract the real input based on the transcripts and compute $f(x_1, \ldots, x_m)$.

Under the assumption that the given PTE protocol satisfies agent privacy, we show computation privacy in the presence of malicious units. We can construct a simulator Sim as follows. The simulator runs a copy of adversary Adv internally which can corrupt any unit U_i. The simulator simulates the transcripts between an honest agent A and $\mathsf{Sim}_{\mathrm{PTE}}$, which simulates a corrupted unit U_i internally; here $\mathsf{Sim}_{\mathrm{PTE}}$ is the simulator for agent privacy in PTE protocol. Now the adversary's view is indistinguishable from the simulated view.

Proof (Proof of theorem 2). Integrity is satisfied in the present of honest units given that the protocol is complete.

Unit privacy is satisfied even in the presence of a malicious agent, we construct a simulator Sim as follows. The simulator runs a copy of the adversary Adv internally which corrupts the agent A; here the corrupted agent's inputs including public key pk, two tables $\mathcal{T}^{(0)}, \mathcal{T}^{(1)}$, and ciphertext $\mathcal{E}^v_{pk}(j)$ are supplied to Sim. The simulator also runs a copy of honest U internally. The simulator supplies pk to U. Whenever receiving the first move message in the protocol from the corrupted A, which include two vectors, totally $2(v-1)$ number of ciphertexts based on pk, the simulator randomly selects $(v-1)$ elements in the ciphertext space $\mathcal{C}_{pk}$ and sends it to A in the name of U. Whenever A outputs a message, the simulator records the message. Based on the DCR assumption, the simulated second move message is distinguishable from the real one, which means the simulated view is computationally indistinguishable from the real view of the adversary.

Agent privacy is satisfied even in presence of the malicious unit. The simulator runs a copy of the adversary Adv internally which corrupts the unit U; here the corrupted unit's input including pk and input bit b are supplied to Sim. The simulator also runs a copy of honest agent A. The simulator runs $(pk, sk) \leftarrow \mathcal{G}(1^k)$, and supplies the pk to both corrupted U and honest A. Whenever the agent is expected to receive inputs, the simulator just supplies it with pk and fakes two meaningful tables $\tilde{\mathcal{T}}^{(0)}, \tilde{\mathcal{T}}^{(1)}$, and a vector of random ciphertexts consistent to the tables. Whenever the agent sends out the first move message in the protocol, the simulator randomly selects two vectors of elements in $\mathcal{C}_{pk}$, and sends them to the corrupted U in the name of A. Under the DCR assumption, the simulated first move message is indistinguishable from the real one, which means the adversary Adv has only negligible probability to distinguish the real view and the simulated view.

Proof (Proof sketch of theorem 3). The theorem holds based on theorem 2, and the fact that the Sigma proof protocol is sound and zero-knowledge.

Given that the Sigma proof is sound unconditionally (the underlying commitment used is unconditionally binding), a malicious unit has to randomize one of two vectors of ciphertexts by multiplying each one in the vector a ciphertext for 0. Otherwise the verifications executed by the honest A will not be valid. This guarantees that the integrity property holds. Given the Sigma proof is zero-knowledge under the DCR assumption, based on the theorem 2, the unit privacy is maintained under the same DCR assumption. Further, agent privacy is same as that in the basic setting, and it holds under the DCR assumption.