Secure Multi-party Computation Minimizing Online Rounds

Seung Geol Choi[1,⋆], Ariel Elbaz[1,⋆], Tal Malkin[1,⋆], and Moti Yung[2]

[1] Columbia University
{sgchoi,arielbaz,tal}@cs.columbia.edu
[2] Google Inc. & Columbia University
moti@cs.columbia.edu

Abstract. Multi-party secure computations are general important procedures to compute any function while keeping the security of private inputs. In this work we ask whether preprocessing can allow low latency (that is, small round) secure multi-party protocols that are universally-composable (UC). In particular, we allow any polynomial time preprocessing as long as it is independent of the exact circuit and actual inputs of the specific instance problem to solve, with only a bound k on the number of gates in the circuits known.

To address the question, we first define the model of "Multi-Party Computation on Encrypted Data" (MP-CED), implicitly described in [FH96, JJ00, CDN01, DN03]. In this model, computing parties establish a threshold public key in a preprocessing stage, and only then private data, encrypted under the shared public key, is revealed. The computing parties then get the computational circuit they agree upon and evaluate the circuit on the encrypted data. The MP-CED model is interesting since it is well suited for modern computing environments, where many repeated computations on overlapping data are performed.

We present two different round-efficient protocols in this model:

- The first protocol generates k garbled gates in the preprocessing stage and requires only two (online) rounds.
- The second protocol generates a garbled universal circuit of size $O(k \log k)$ in the preprocessing stage, and requires only one (online) round (i.e., an obvious lower bound), and therefore it can run asynchronously.

Both protocols are secure against an active, static adversary controlling any number of parties. When the fraction of parties the adversary can corrupt is less than half, the adversary cannot force the protocols to abort.

The MP-CED model is closely related to the general Multi-Party Computation (MPC) model and, in fact, both can be reduced to each other. The first (resp. second) protocol above naturally gives protocols for three-round (resp. two-round) universally composable MPC secure against active, static adversary controlling any number of parties (with preprocessing).

Keywords: Computing with Encrypted Data, Multi-Party Computation, Public key Cryptography, Cryptographic Protocols, Universal Composition.

[⋆] Supported in part by NSF Grants CCF-0347839, CNS-0716245 and CNS-0831094.

M. Matsui (Ed.): ASIACRYPT 2009, LNCS 5912, pp. 268–286, 2009.

1 Introduction

Secure Multi-party Computation (MPC). Protocols for MPC enable a set of parties to correctly evaluate a function such that no information about the private inputs of the parties is revealed, beyond what is leaked by the output of the function. This notion was first presented by Yao [Y86] for the two-party case, and by Goldreich et al. [GMW87] for the multi-party case. However, implementations for MPC are notoriously inefficient. Many protocols implementing them have delays associated with the depth of the circuit and even constant round protocols produce very long delays. The question that we want to settle in this work is whether one can use preprocessing computation in order to "be ready" once the inputs and the actual circuit (problem) to compute on are given. Note that the world of computing is transforming into "cloud services" where parties can "rent" computational resources. Thus, it may make sense to perform a lengthy preprocessing in the background, with no specific input and problem to solve in mind, just as a preparation. To this end cloud resources can be employed on behalf of users, and massive computations and communication can be performed. Then in the online stage once the input is given and the circuit determined, it can be performed much faster given the preprocessing. As long as at least one of the servers in the cloud is not corrupted, the correctness and privacy of the online stage computation is guaranteed.

We consider the following variation on secure multi-party computation, called *multi-party computing with encrypted data (*MP-CED*)*: (1) The computing parties publish a shared public key, and hold shares of the matching private key. (2) The parties also know some bound on the circuit size that they will be required to compute securely. The parties then perform a preprocessing stage. For this stage too, we may try to minimize the parties' work and computation rounds, but this is not the main goal, which is the efficiency of the on-line stage. (3) The input distribution is a database of encrypted data that can be published by many parties (not necessarily those taking part in the computation); i.e., think about the parties as a service (like the census bureau) computing on behalf of a larger population. (4) The concrete computation circuit (or circuits) is given, and the input to use from the database (their indices in the database) are determined. Then and only then (5) the parties are engaged in a short computation to achieve the task and produce the output while protecting the private data. Note that the input database may be reused for many computations.

We remark that our model is somewhat related to a multi-party extension of the model by Rivest, Adleman and Dertouzos [RAD78]. They put forth a scenario for secure computation over database of encrypted data, called *Computing with Encrypted Data* (CED). This model is highly attractive since it represents the case where a database is first collected and maintained and only later a computation on it is decided upon and executed (e.g., data mining and statistical database computation done over the encrypted database). We discuss the encrypted data model and the multi-party version here, and in fact show that MP-CED and MPC can be reduced to each other (shown in Section 3.3).

1.1 Motivation

We consider protocols in the universal composability (UC) framework introduced by Canetti [C01]. UC secure protocols remain secure even when executed concurrently with arbitrary other protocols running in some larger network, and can be used as subroutines of larger protocols in a modular fashion.

Round-Efficient Protocols with Preprocessing. Round complexity is an important criterion for the efficiency of an MPC protocol. A long line of work, including [BMR90, IK00, GIKR01, DI05, DI06, DIK^{+}08], focused on reducing both the round complexity and communication complexity.

Also, it is known that UC secure computation of general functions is not possible in the plain model in the case of honest minority. In particular, UC secure two-party computation of a wide class of functionalities was ruled out by [CF01, CKL03]. To circumvent these impossibility results, it is common to assume some pre-computation setup, and the most common assumption is that a common reference string (CRS) is made available to the parties before the computation. Canetti et al. [CLOS02] showed that (under suitable cryptographic assumptions) a CRS suffices for UC secure MPC of any well-formed functionality.

In our work, we consider stronger relaxation on the setup, called general preprocessing [DI05][1], in which the parties perform some work as long as it is independent of the inputs and the circuit for which the actual computation is to be done later. The main motivation for this model is to reduce the amount of work during the execution of the protocol beyond a preprocessing phase.

Considering the two aspects above, we ask the following natural question:

> *Allowing any polynomial time preprocessing (in some input parameter) before the circuit (whose size is bound by the same input parameter) and the inputs are known, is there a very small constant round protocol?*

1.2 Our Results

We address the aforementioned question affirmatively by constructing two different round-efficient protocols for MP-CED, which we call $\mathcal{P}_1$ and $\mathcal{P}_2$. Both protocols can be naturally transformed into round-efficient protocols for MPC (c.f. Section 3.3). Each protocol has its own advantage depending on the following parameters:

1. round complexity in the online stage (our major concern),
2. round complexity in the preprocessing stage, and
3. the number of gates constructed throughout the protocol.

In terms of online round complexity, protocol $\mathcal{P}_1$ is “two rounds” whereas that of protocol $\mathcal{P}_2$ is “one round” (which is optimal, since even non-secure computation need to collect the data and it takes one round). There are some cases, however, in which the preprocessing round complexity of $\mathcal{P}_1$ is better, under some efficiency considerations. We use general constant-round MPC protocols [IPS08] for the preprocessing stage in $\mathcal{P}_2$, whereas in $\mathcal{P}_1$ we can use the protocol given in Appendix A, which requires *exactly* $2n$ rounds. When n is small enough, preprocessing in $\mathcal{P}_1$ can be more round-efficient

[1] Preprocessing in [DI05] is independent only of the inputs (it depends on the circuit to be evaluated), whereas we require preprocessing to be independent both of the circuit and of the inputs.

(when n is large, a general MPC protocol can be used in $\mathcal{P}_1$, too). Also, the number of gates constructed in $\mathcal{P}_2$ is larger than that in $\mathcal{P}_1$. To evaluate a circuit with up to k gates, $\mathcal{P}_1$ constructs k garbled gates in the preprocessing stage, as explained below. In contrast, $\mathcal{P}_2$ generates a universal circuit [V76] in the preprocessing stage, which is later used (in the online stage) to evaluate a given circuit. The smallest known universal circuit that can evaluate a circuit with k gates has $O(k \log k)$ gates [KS08]. We overview the two protocols in the following.

First Protocol ($\mathcal{P}_1$). In a big picture, we follow the framework of Yao's garbled circuit technique. However, the main difference is that, in our protocol, garbling is done *on the individual gate level* so that this procedure can be executed in the preprocessing level independently of the circuit to be given and computed later. In the online stage, construction of wires between gates according to the given circuit is performed.

- In the preprocessing stage, the parties generate a 'garbled' truth table for each individual gate. Truth tables are for NAND gates, and they have four rows and three columns – left-input, right-input, and output. Each row is randomly shuffled, and each element is an encryption of Boolean value. We emphasize that no party knows anything more than the fact that it's a randomly shuffled encrypted table for NAND.

  In addition, a fresh pair of public key and (encrypted) private key is generated for each row. This key is used for constructing encrypted wiring information in the online stage, when the circuit is given.
- In the online stage, given the encrypted data and a circuit, the computing parties 'connect' truth tables by adding *wiring information*. The wiring information tells, given two tables T_{pred}, T_{succ} according to the topology of the circuit, which row of T_{pred}'s output column is equal to which row of T_{succ}'s input column. We note that this information should be carefully revealed; otherwise, the adversary may try computing different rows of the truth tables using the wirings, and may learn more than is allowed. In fact, during the computation (online stage), exactly one row's wirings for each table should be revealed.

  To enable such wirings we introduce *Multi-Party Conditional Oblivious Decryption Exposure (*M-CODE*)* (in Section 2), which is a multi-party extension to the CODE functionality, introduced in [CEJ$^+$07] for the two party case. M-CODE assumes a group of parties share a secret key x of a public key y. Three ciphertexts c_{out}, c_{in}, c_{key} — all encrypted under y — and a new public key z are given as input. For $\ell \in \{out, in, key\}$, let m_ℓ be the plaintext encrypted in c_ℓ. If m_{out} equals m_{in}, M-CODE outputs $E_z(m_{key})$. Otherwise, M-CODE chooses a random value r and outputs $E_z(r)$. The computing parties use M-CODE such that, for each row of a truth table, the three ciphertexts of the M-CODE are (1) output value of the previous table (2) the input value of this row and (3) the secret key for this row. We refer the reader to Section 3.1 for more details.

  With two round implementation of M-CODE for ElGamal encryption, we obtain a two-round protocol for MP-CED and a three-round protocol for MPC.

Theorem 1. *Assuming the DDH assumption holds, protocol $\mathcal{P}_1$ is a two-round UC secure protocol for* MP-CED *in the $\mathcal{F}_{zk}$ hybrid — and, thereby three-round UC secure protocol for* MPC *in the $\mathcal{F}_{zk}$ hybrid in the general preprocessing model — against an active and static adversary as long as at most $t < n$ computing parties are corrupted.*

The protocols manipulate linear number of gates in the circuit size. Furthermore, if $t < n/2$ parties are corrupted, $\mathcal{P}_1$ is robust against abort.[2]

Second Protocol ($\mathcal{P}_2$). Protocol $\mathcal{P}_2$ follows Yao's garbled technique more closely than $\mathcal{P}_1$. However, the circuit that is to be garbled is a universal circuit [V76, KS08] to maintain independence of the circuit to be given. Optimal round complexity in the online stage is achieved by putting a simple constraint on the input-layer labels in the garbled circuit and by employing the multiplicative homomorphism of ElGamal encryption. As in the first protocol, a group of parties share a secret key x of a public key y.

- In the preprocessing stage, the parties generate a garbled circuit [Y86] of a universal circuit C_U, with some special restrictions on keys of input wires. In the garbled circuit C_U, there are two keys w_0^i and w_1^i for each wire i, where w_b^i corresponds to the wire carrying bit b (see Section 3.2 for more detail). The special restriction on input wires is that $w_1^i/w_0^i = h$ for a random global value h unknown to any party. The two keys can be constructed by picking w_0^i uniformly at random and letting $w_1^i = h \cdot w_0^i$. In addition to the garbled circuit of C_U, the following encryptions are generated: (1) the encryption $E_y(h)$ and (2) $E_y(w_0^i)$ for each input wire i. Construction of a garbled circuit along with aforementioned encryptions — i.e., $E_y(h)$ and $E_y(w_0^i)$'s — can be performed using a constant-round UC secure protocols for general MPC [KOS03, IPS08]. Input contribution of a bit 0 is done by $E_y(h^0)$, and for a bit 1, re-encrypted $E_y(h^1)$ is used via homomorphism.
- In the online stage, for each input wire i where a bit b is the contributed input for the wire, computing parties obtain w_b^i. The encryption $E_y(w_b^i)$ can be obtained via homomorphism given the encrypted input $c_i = E_y(h^b)$, giving $E_y(w_0^i) \cdot c_i = E_y(w_0^i h^b) = E_y(w_b^i)$, since $w_1^i = h \cdot w_0^i$. Now parties obtains the key w_b^i for each input wire i using threshold decryption and can locally evaluate the garbled circuit. Note that w_b^i does not leak any information on b since it's randomly distributed (with w_{1-b}^i hidden).

Theorem 2. *Assuming the DDH assumption holds, protocol $\mathcal{P}_2$ is a one-round UC secure protocol for* MP-CED *in the $\mathcal{F}_{zk}$ hybrid – and, thereby two-round UC secure protocol for* MPC *in the $\mathcal{F}_{zk}$ hybrid in the general preprocessing model – against an active and static adversary as long as at most $t < n$ computing parties are corrupted. The protocol processes $k \log k$ gates where k is the circuit size. Furthermore, if $t < n/2$ parties are corrupted, $\mathcal{P}_2$ is robust against abort.*

1.3 Related Work

Round Complexity. Beaver et al. [BMR90] showed the first MPC protocol that required constant (but large) number of rounds, and Damgård and Ishai [DI05] presented the first adaptively UC secure protocol that achieves two rounds in the (linear) preprocessing model when the number of malicious parties $t < n/5$ and some higher constant rounds when $t < n/2$. Recently, Ishai et al. constructed UC secure protocol with malicious majority in the OT hybrid model running in (large) constant rounds [IPS08] (see Fig 1).

[2] Instantiation of protocol $\mathcal{P}_1$ (in particular, key setup in the preprocessing) is parameterized by t. Therefore protocol $\mathcal{P}_1$ is not a 'best-of-both-worlds' protocol [IKLP06]. This is true of $\mathcal{P}_2$, too.

MPC	circuit	rounds	security	♯corr
[KOS03]	B	O(1)	St	$t < n$
[DI05]	B	2	Ad	$t < n/5$
[DI05, DI06]	B	O(1)	Ad	$t < n/2$
[DIK$^+$08]	B	O(1)	Ad	$t < n/2$
[IPS08]	B	O(1)	Ad	$t < n$
$\mathcal{P}_1$	B	3	St	$t < n$
$\mathcal{P}_2$	B	2	St	$t < n$

MP-CED	circuit	rounds	security	♯corr
[CDN01]	Ar	$O(d)$	SA, St	$t < n/2$
[JJ00]	B	$O(n + d)$	SA, St	$t < n/2$
[DN03]	Ar	$O(d)$	UC, Ad	$t < n/2$
$\mathcal{P}_1$	B	2	UC, St	$t < n$
$\mathcal{P}_2$	B	1	UC, St	$t < n$

♯corr = number of corrupted parties, B = Boolean, Ar = Arithmetic, St = static, Ad = adaptive, SA = stand-alone

Fig. 1. UC Secure Constant-Round MPC Protocols (Left) and MP-CED Protocols (Right). We denote by d the depth of a given circuit, by n the number of parties, and by t the number of corrupted parties. $\mathcal{P}_1$ and $\mathcal{P}_2$ denote the protocols proposed here. Here the column 'rounds' means the number of rounds in the online stage.

For the two-party setting, which is a special case of MPC, Katz and Ostrovsky [KO04] showed that it's impossible to construct a secure protocol running in four rounds using enhanced trapdoor permutation (eTDP) or homomorphic encryption in a black-box manner in the plain model, and they constructed a five-round protocol. To overcome this lower bound, Horvitz and Katz [HK07] used CRS to construct a UC secure two-party protocol in two rounds. Nielsen and Orlandi [NO09] gave a two party protocol using a cut-and-choose approach. In a big picture, their idea is somewhat similar to ours: after many garbled gates are generated, they are connected to each other according to the circuit to be evaluated.

In the (non-UC) stand-alone setting, the work of [IK00, AIK05] gave a general non-interactive reduction of any n-party functionality computed by a polynomial size Boolean circuit into a (possibly randomized) functionality of degree-3 over $GF(2)$. Combining this reduction with any secure protocol with malicious majority (for example, [GMW87]) leads to round-efficient protocols in the stand-alone setting.

MP-CED. Some nontrivial instantiations for CED were shown, originating with Sander et al. [SYY99], who gave a protocol for circuits in NC^1. Beaver [B00] extended this result to accommodate any function in NLOGSPACE [BL96]. Recently, Gentry presented a construction for any polynomial size circuit by showing doubly-homomorphic encryption scheme from ideal lattices [G09], however it is not yet clear if this can give efficient protocols for MP-CED (see discussion in Section 3.3).

MP-CED was also considered by Franklin and Haber [FH96] and the subsequent works [JJ00, CDN01, DN03]. In their works, after a threshold encryption key is established, each party broadcasts the encryption of its input, and the parties evaluate the circuit on the encrypted data. However, they do not explicitly treat the setting as a unique model for MP-CED, with a specific setup state that is independent of the inputs and the circuits to be computed, and do not consider *input separation* – inputs can be contributed by parties that do not take part in the computation. Note that all these previous works in the model dealt with the two party case, which we extend herein to the multi-party case.

The protocol given by Cramer et al. [CDN01] computes an arithmetic circuit and achieves security in the case of honest majority, but the number of rounds is linear in the depth of the circuit. A UC adaptively secure protocol with the same round complexity was given by [DN03]. Jackobson and Juels [JJ00] use mix-and-match approach to compute on encrypted data, but their approach requires even more rounds (linear in the sum of the depth of the circuit and the number of parties). Figure 1 lists these previous works, in some relations to our protocols (while concentrating on on-line rounds, and omitting some of the advantages our results has beyond the table).

2 Preliminaries

For any integer t, let $[t] = \{0, 1, \ldots, t-1\}$. Let k be a security parameter. We choose a cyclic group $\mathcal{G}_g^q$ of order $q \approx 2^{\mathsf{k}}$ with a generator g where the DDH problem [DH76] is hard. For example, $\mathcal{G}_g^q$ can be a subgroup of order q of a multiplicative group Z_p^* for a safe prime $p = 2q + 1$, i.e., $\mathcal{G}_g^q = \{g^0, g^1, \ldots, g^{q-1}\} \pmod p$. We assume $\mathcal{G}_g^q$ is known in advance.

ElGamal Encryption. ElGamal encryption [E85] is semantically secure under the DDH assumption over $\mathcal{G}_g^q$ [TY98]. *The key generation algorithm* generates a public/secret key pair (y, x) where $x \in_R [q]$ and $y = g^x$. *Encryption* of a message $m \in \mathcal{G}_g^q$ under a public key y, denoted by $E_y(m)$, is (g^r, my^r) where $r \in_R [q]$. *Decryption* of a ciphertext $c = (\alpha, \beta)$ with the secret key x, denoted by $D_x(c)$, is β/α^x.

Homomorphism. Multiplication of two ciphertexts $E_y(m_1) = (g^{r_1}, m_1 y^{r_1})$ and $E_y(m_2) = (g^{r_2}, m_2 y^{r_2})$ is defined as $(g^{r_1+r_2}, m_1 m_2 y^{r_1+r_2})$, which shows the homomorphism of ElGamal encryption (i.e., $E_y(m_1) \cdot E_y(m_2) = E_y(m_1 \cdot m_2)$). In addition, encryption keys are also homomorphic in the sense that given key pairs $\{(y_i = g^{x_i}, x_i)\}_i$, the pair $(\prod_i y_i, \sum_i x_i)$ is a valid key pair. When two ciphertexts encrypt the same message, we denote $c_1 \equiv c_2$.

Zero-Knowledge Proofs of Knowledge (ZK-PoK). A proof of knowledge is a proof for a relation R, in which the prover convinces the verifier that an instance is in the language, and also that *the prover knows a witness for this instance*. We will use standard notation to denote proofs of knowledge related to discrete log. For example, $PK\{b : a = g^b\}$ denotes a proof of knowledge where the prover convinces the verifier that she knows the value of b, such that $a = g^b$, when a is known to both.

In the common reference string (CRS) model, we can use non-interactive zero-knowledge proofs (NIZK) due to De Santis et al. [SCO$^+$01] (see the discussion in [CLOS02, Section 6]) which is UC-secure [C01]. In the random oracle model (ROM), the above proof systems can be efficient NIZK using the standard Fiat-Shamir technique [FS86] combined with OR proofs of Σ-protocols [CDS94].

Secret Sharing [S79, F87]. A secret sharing scheme allows a secret $s \in [q]$ to be shared among n parties, such that a threshold of $t+1$ parties can recover the secret, whereas any smaller set of parties can not learn anything about the secret. In Shamir's secret sharing scheme, the shares are values of a degree-t polynomial, and the secret is the free coefficient of the polynomial.

We show below how the parties can share and recover the secret s. Moreover, the parties may choose to recover d^s for some $d \in \mathcal{G}_g^q$, or an ElGamal encryption of d^s (without learning anything about the secret s).

- *Sharing:* A dealer chooses at random a degree t polynomial $Q(x) := s + a_1 x + \cdots + a_t x^t$, where the free coefficient is the secret s. The share of party P_i is $s_i = Q(i)$.
- *Recovering s:* Let T be a set of $t+1$ parties. They evaluate $Q'(0) = \sum_{i \in T} s_i L_i(0)$ to recover s, where L_i is a Lagrangian on the points in T.[3]
- *Recovering an exponentiation d^s:* Similar to above, the parties can evaluate $d^s = d^{Q'(0)} = d^{\sum_{i \in T} s_i L_i(0)} = \prod_{i \in T} d^{s_i L_i(0)}$, using only $\{d^{s_i}\}_{i \in T}$.
- *Recovering $E_y(d^s)$:* Using multiplicative homomorphism of ElGamal, the parties evaluate $E_y(d^s) = E_y(d^{Q'(0)}) = \prod_{i \in T} E_y(d^{s_i L_i(0)}) = \prod_{i \in T} E_y(d^{s_i})^{L_i(0)}$, using only $\{E_y(d^{s_i})\}_{i \in T}$.

Multi-party Conditional Oblivious Decryption Exposure (M-CODE). We introduce *Multi-Party Conditional Oblivious Decryption Exposure (*M-CODE*).* M-CODE assumes a group of parties share a secret key x of a public key y. Three ciphertexts c_{out}, c_{in}, c_{key} — all encrypted under y — and a new public key z are given as input. For $\ell \in \{out, in, key\}$, let m_ℓ be the plaintext encrypted in c_ℓ. If m_{out} equals m_{in}, M-CODE outputs $E_z(m_{key})$. Otherwise, M-CODE chooses a random value r and outputs $E_z(r)$. A variant of this functionality for the two party case was initially introduced by [CEJ+07]. The intuitive idea is to generate a ciphertext that encrypts m_{key} multiplied by $(m_{out}/m_{in})^r$ for a random r. If $m_{in} = m_{out}$, then the output would be m_{key}. We assume party P_i has x_i, all the parties know $c_{out}, c_{in}, c_{key}, z, (y, y_1 = g^{x_1}, \ldots, y_n = g^{x_n})$, and let $c_{out} = E_y(m_{out}) = (\alpha, \beta)$,$c_{in} = E_y(m_{in}) = (\gamma, \delta)$, $c_{key} = E_y(m_{key}) = (\lambda, \mu)$. The protocol for M-CODE proceeds as follows:

1. Each party P_i chooses $e_i \in_R [q]$, and computes $\epsilon_i = (\alpha/\gamma)^{e_i}$, $\zeta_i = (\beta/\delta)^{e_i}$, $\pi_i = PK\{e_i : \epsilon_i = (\alpha/\gamma)^{e_i}$, and $\zeta_i = (\beta/\delta)^{e_i}\}$,, and broadcasts $(\epsilon_i, \zeta_i, \pi_i)$.
2. Let $\epsilon = \prod_{i \in S_1} \epsilon_i$ and $\zeta = \prod_{i \in S_1} \zeta_i$ where S_1 is the set of parties which sent valid messages. Each party P_i chooses r_i randomly and computes $d_i = (d_{i1}, d_{i2}) = E_z\left((\epsilon\lambda)^{x_i}\right)$ and $\psi_i = PK\Big\{(r_i, x_i) \ : \ d_{i1} = g^{r_i} \ , \ d_{i2} = z^{r_i}(\epsilon\lambda)^{x_i} \ , \ y_i = g^{x_i}\Big\}$, and broadcasts (d_i, ψ_i).
3. Let S_2 be the set of parties that sent valid messages in steps 1 & 2. If $|S_2| \le t$, then the protocol aborts. Each party P_i, using the homomorphic multiplication, computes $d = (d_1, d_2) = E_z\left((\epsilon\lambda)^x\right) = \prod_{j \in S_2} d_j^{L_j(0)}$ where $L_j(\cdot)$ is a Lagrangian on the indices in S_2. P_i uses homomorphic operations to compute $E_z\left(\tilde{m_{key}}\right) = (1/d_1, \zeta\mu/d_2)$, which is
$$E_z\left(\zeta\mu/(\epsilon\lambda)^x\right) = E_z\left(\left(\frac{\beta/\alpha^x}{\delta/\gamma^x}\right)^e \cdot (\mu/\lambda^x)\right) = E_z\left(\left(\frac{m_{out}}{m_{in}}\right)^e \cdot m_{key}\right),$$
where $e = \sum_{i \in S_1} e_i$.

[3] Lagrangian L_i on the points in T is a degree t polynomial such that $L_i(x) = 1$ if $x = i$ and $L_i(x) = 0$ if $x \in T$ and $x \neq i$. The polynomial $Q'(x) = \sum_{i \in T} s_i L_i(x)$ is a degree t polynomial that goes through the points $(i, s_i)_{i \in T}$, and thus must be $Q(x)$.

3 Multi-party Computing with Encrypted Data

We assume the circuit C of interest is normalized: all intermediate gates are NAND gates, and output gates are IDENTITY gates[4]. We can easily attain this circuit by adding another layer of IDENTITY gates on top of a circuit that consists of NAND gates.

3.1 First Protocol ($\mathcal{P}_1$)

In the first protocol, called $\mathcal{P}_1$, each gate is garbled, and then the computing parties 'connect' gates by adding *wiring information* using M-CODE.

Preprocessing Stage. The first step is to establish a global public key y for ElGamal encryption. The computing parties have shares of the corresponding secret key x. Once the public key is established, the next step is to generate truth tables for individual gates. The columns of input, output, and intermediate gates differ slightly, as can be seen in Figure 2 which shows the structure of truth tables.

1. Input and Output. These are encrypted with the global public key y.
2. Placeholders for the wiring information. This connects a row of the truth table to matching rows in successor gates.
3. The columns PK and SK contain a random ElGamal key pair, where the private key is encrypted under the global public key y (and the wiring information is encrypted using the secret keys in SK).
4. For output gates, ciphertexts in column Final encrypt the same plaintexts as ciphertexts in column In.

During the preprocessing stage, the parties can generate polynomial number of garbled gates, that can later be used for evaluating circuits. Therefore it suffices to know a bound on the sizes of circuits to be evaluated later. Preprocessing can be done in constant number of round using general MPC protocols [KOS03, IPS08]. If the number of computing parties is small, it can be done explicitly in $2n$ rounds, where n is the number of computing parties, using the protocol in Appendix A.

Input contribution is performed by publishing a ciphertext $c = (c_1, c_2) = E_y(g^b)$ for an input $b \in \{0,1\}$. This can be done securely by adding $PK\{r : (c_1 = g^r,\ c_2 = y^r) \text{ or } (c_1 = g^r,\ c_2 = gy^r)\}$.

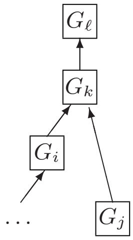
Online Stage: Generation of Wires Between Garbled Gates. In Figure 2, G_i is the left predecessor of G_k. The connection between the two gates should be established through some "wiring" such that during the computation the output of G_i can be propagated to the left input of G_k. So, rows of T_i with output value $b \in \{0,1\}$ should be connected to rows of T_k with left input value b.

Requirements for Wiring. In our protocol, the following conditions are considered in generating wires.

[4] An IDENTITY gate has single input bit (wire) and output bit, and it copies the input bit value to its output.

T_ℓ	In	PK	SK	Final
1	$E[1]$	$pk_{\ell 1}$	$E_y(sk_{\ell 1})$	$E_{pk_{\ell 1}}(g^1)$
2	$E[0]$	$pk_{\ell 2}$	$E_y(sk_{\ell 2})$	$E_{pk_{\ell 2}}(g^0)$

T_k	In$_L$	In$_R$	Out	PK$_L$	SK$_L$	PK$_R$	SK$_R$	Wires$_{[k \to \ell]}$
1	$E[1]$	$E[1]$	$E[0]$	pk_{k1L}	$E_y(sk_{k1L})$	pk_{k1R}	$E_y(sk_{k1R})$	$E_{pk_{k1L} \cdot pk_{k1R}}(*, sk_{\ell 2})$
2	$E[1]$	$E[0]$	$E[1]$	pk_{k2L}	$E_y(sk_{k2L})$	pk_{k2R}	$E_y(sk_{k2R})$	$E_{pk_{k2L} \cdot pk_{k2R}}(sk_{\ell 1}, *)$
3	$E[0]$	$E[1]$	$E[1]$	pk_{k3L}	$E_y(sk_{k3L})$	pk_{k3R}	$E_y(sk_{k3R})$	$E_{pk_{k3L} \cdot pk_{k3R}}(sk_{\ell 1}, *)$
4	$E[0]$	$E[0]$	$E[1]$	pk_{k4L}	$E_y(sk_{k4L})$	pk_{k4R}	$E_y(sk_{k4R})$	$E_{pk_{k4L} \cdot pk_{k4R}}(sk_{\ell 1}, *)$



T_i	In$_L$ In$_R$	Out	PK$_L$ SK$_L$ PK$_R$ SK$_R$	Wires$_{[i \to k]}$
1	$\cdots$	$E[1]$	$\cdots$	$E_{pk_{i1}}(sk_{k1L}, sk_{k2L}, *, *)$
2	$\cdots$	$E[0]$	$\cdots$	$E_{pk_{i2}}(*, *, sk_{k3L}, sk_{k4L})$
3	$\cdots$	$E[1]$	$\cdots$	$E_{pk_{i3}}(sk_{k1L}, sk_{k2L}, *, *)$
4	$\cdots$	$E[1]$	$\cdots$	$E_{pk_{i4}}(sk_{k1L}, sk_{k2L}, *, *)$

T_j	Out	Wires$_{[j \to k]}$
1	$E[1]$	$(sk_{k1R}, *, sk_{k3R}, *)$

Fig. 2. Garbled Truth Tables for the Gates (G_i, G_j, G_k, G_ℓ)**.** The topology of the gates is given on the right. G_j is an input gate, G_ℓ is an output gate, and G_i, G_k are intermediate gates. Table T_x is the truth table describing gate G_x. y is the global public key. Each row of an intermediate truth table has two sets of (secret, public) keys, and contains the wiring information, "connecting" it to the next gate, encrypted using these two keys. $E[0]$ and $E[1]$ are $E_y(g^0)$ and $E_y(g^1)$ respectively. In table T_i, $pk_{i1} = pk_{i1L} \cdot pk_{i1R}$, and $pk_{i2}, \ldots, pk_{i4}$ are defined similarly. In the Wires columns, $E(a, b, c, d)$ denotes concatenation of $E(a), \ldots, E_(d)$.

- (Encrypting the Wiring Information.) The wiring information, except wirings connecting an input gate to an intermediate gate, should be encrypted. Public wiring may help the (even semi-honest) adversary to learn more information than the output of C. Therefore, it is encrypted with the public key stored in columns PK$_L$ and PK$_R$.
- (Conditional Exposure of Wiring Information.) For the computation to proceed, the protocol should reveal the wiring information for the rows along the computational path. In the beginning, wirings from input gates is public. Along the computational path, on each gate, exactly one row should allow decryption of the wiring information.
- (Oblivious Generation of Wiring Information.) The wiring information are added to garbled gates after they are built. It is essential that, even if the truth table is encrypted and shuffled, the parties should still be able to add the wiring information.

Computation of a Circuit Using Wires. Let $T_i[a][b]$ denote the element located at column a and row b in T_i. The column Wires contains wiring information, and we denote the column Wires from T_i to T_k by Wires$_{[i \to k]}$[5]. Looking at the column Wires alone, Wires(v) denotes the vth row of this column in the plaintext form. For example, Wires$_{[i \to k]}(2) = (*, *, sk_{k3L}, sk_{k4L})$ in Figure 2. We also use Wire(v, w) to denote the wth element of Wires(v). If Wire$_{[i \to k]}(v, w) \neq *$, it means that $T_i[\text{Out}][v] \equiv T_k[\text{In}][w]$. In Figure 2, for example, we have Wire$_{[i \to k]}(2, 3) \neq *$ because $T_i[\text{Out}][2] \equiv T_k[\text{In}_L][3] \equiv E[0]$.

This wiring information helps the circuit computation to proceed correctly. The computation proceeds in order from input gates to output gates. In Figure 2, for example, if

[5] If G_i has another outgoing wire, say to G_m, T_i will have another column Wires$_{[i \to m]}$.

Connecting the Gates: Fill in Wires **Columns.**

- For every $\mathsf{Wire}_{[i \rightarrow k]}(v, w)$ of an intermediate gate T_i, run M-CODE for $c_{out} = T_i[\mathsf{Out}][v]$, $c_{in} = T_k[\mathsf{In}][w]$ and $c_{key} = T_k[\mathsf{SK}][w]$, with the key $z = T_i[\mathsf{PK}_L][v] \cdot T_i[\mathsf{PK}_R][v]$.
- For every $\mathsf{Wire}_{[j \rightarrow k]}(v, w)$ of an input gate T_j, run M-CODE for $c_{out} = T_j[\mathsf{Out}][v]$, $c_{in} = T_k[\mathsf{In}][w]$ and $c_{key} = T_k[\mathsf{SK}][w]$, with the trivial key $z = g^0$.

Depending on the circuit topology, the subscript of a column may differ (e.g., In_L, In_R, or In).

Local Computation. Each party computes the output of C using the M-CODE transcripts on the input gates.

Fig. 3. Online Stage of $\mathcal{P}_1$

row 2 of T_i and row 1 of T_j are on the computation path, then row 3 of T_k is also on the computation path because $w = 3$ is the only row where $\mathsf{Wire}_{[i \rightarrow k]}(2, w) \neq *$ and $\mathsf{Wire}_{[j \rightarrow k]}(1, w) \neq *$.

Constructing Wires. We implement each $\mathsf{Wire}_{[i \rightarrow k]}(v, w)$ using a M-CODE transcript for $c_{out} = T_i[\mathsf{Out}][v]$, $c_{in} = T_k[\mathsf{In}][w]$, $c_{key} = T_k[\mathsf{SK}][w]$, and $z = T_i[PK][v]$[6]. This directly satisfies the requirements of encrypted wiring and oblivious wiring generation. Conditional exposure is achieved by executing M-CODE protocols in the input layer with a trivial public key $z = 1$, so that the wiring information in the input layer is known to every party.

The description of $\mathcal{P}_1$ can be found in Figure 3. Running the online stage takes two rounds. The communication complexity of $\mathcal{P}_1$ is $\mathcal{O}(n\mathsf{k}|C|)$ (plus the NIZK, if we assume the CRS case) where $|C|$ is the size of the circuit.

3.2 Second Protocol ($\mathcal{P}_2$)

The idea of $\mathcal{P}_2$ is that in a preprocessing stage, the parties generate a garbled circuit, using Yao's technique, of a universal circuit. The garbled circuit has a restriction on the keys of input wires, that allows the online computation to take only one round in our model, as opposed to the two-round OT based approach of Yao. The preprocessing stage can be done in constant number of rounds, using general MPC protocols [KOS03, IPS08].

Preprocessing Stage: Garbling Universal Circuit. The first step is to establish a global public key y for ElGamal encryption. The computing parties have shares of the corresponding secret key x. In contrast to protocol $\mathcal{P}_1$, however, here, ElGamal encryption is used only for input layer.

Next, a garbled circuit for *universal circuit* is generated, using Yao's garbled circuit technique [Y82]. In the generation procedure, for each wire i, two random keys, w_0^i and w_1^i are generated. The key w_0^i (resp., w_1^i) represents 0 (resp., 1) for wire i. For each gate G_j, a truth table T_j is generated. In each table, a private key encryption (denoted

[6] Depending on the circuit topology, if this is a left input or right input to the gate, the pair (c_{in}, c_{key}) may also be $(T_b[\mathsf{In}_L][w], T_b[\mathsf{SK}_L][w])$ or $(T_b[\mathsf{In}_R][w], T_b[\mathsf{SK}_R][w])$.

T_j	
1	$W_0^j := E_y(w_0^j)$

T_i	
1	$\breve{E}_{(\cdot)}(\breve{E}_{(\cdot)}(w_1^i))$
2	$\breve{E}_{(\cdot)}(\breve{E}_{(\cdot)}(w_0^i))$
3	$\breve{E}_{(\cdot)}(\breve{E}_{(\cdot)}(w_1^i))$
4	$\breve{E}_{(\cdot)}(\breve{E}_{(\cdot)}(w_1^i))$

T_k	
1	$\breve{E}_{w_1^i}(\breve{E}_{w_1^j}(w_0^k))$
2	$\breve{E}_{w_1^i}(\breve{E}_{w_0^j}(w_1^k))$
3	$\breve{E}_{w_0^i}(\breve{E}_{w_1^j}(w_1^k))$
4	$\breve{E}_{w_0^i}(\breve{E}_{w_0^j}(w_1^k))$

T_ℓ	
1	$\breve{E}_{w_1^k}(1)$
2	$\breve{E}_{w_0^k}(0)$

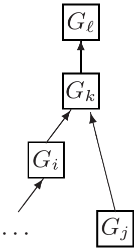


Fig. 4. Garbled Truth Tables for the Gates (G_i, G_j, G_k, G_ℓ)**.** The topology of the gates is given on the right. G_j is an input gate, G_ℓ is an output gate, and G_i, G_k are intermediate gates. Table T_x is the truth table describing gate G_x. y is the global public key. Encryption $\breve{E}$ is a private key encryption based on pseudorandom function with efficient verifiable range [LP09].

$\breve{G}, \breve{E}, \breve{D}$) with efficiently verifiable range (based on pseudorandom function) is used [LP09][7]. Figure 4 shows the structure of the garbled circuit.

- Recall that we assume all output gates are identity gates, with only one incoming wire and only two rows in the corresponding truth table. Each row encrypts the Boolean value represented by the corresponding wire, and the rows are randomly shuffled. An example is given in Figure 4: in the first row of table G_ℓ, the input value is 1 (the key w_1^k represents 1), and it encrypts 1, which is the output value of this row.
- For all other gates, each gate has two incoming wires and four rows. Each row encrypts a key for the outgoing wire, which represents the appropriate Boolean value of NAND of the incoming wires' values, and the rows are randomly shuffled. For example, in G_k of Figure 4, the first row encrypts w_0^k, representation of 0 for wire k, since NAND of the values that the keys of the incoming wires represent (i.e., the value 1 represented by w_1^i in wire i, and the value 1 represented by w_1^j in wire j) is 0.

To construct a secure protocol for MP-CED, we depart from the traditional Yao garbed circuit technique, by giving restriction on input wires.

- A random element $h \in \mathcal{G}_g^q$ is chosen, which no party knows, and $H = E_y(h)$ is published. We emphasize that H is generated *once and for all*. In other words, every instance of garbled universal circuit can use the same H.
- For input wire j, two keys $w_0^j, w_1^j \in \mathcal{G}_g^q$ are randomly generated, conditioned on $w_1^j = h \cdot w_0^j$. Only the encryption of the first key, $W_0^j := E_y(w_0^j)$ is published.

Since we garble a universal circuit, it suffices to know a bound on the sizes of circuits to be evaluated later. A universal circuit of size $O(k \log k)$ can accept circuits of size k as inputs [KS08].

Input contribution is performed such that for input $b \in \{0, 1\}$, a ciphertext $c = (c_1, c_2) = E_y(h^b)$ is published.

[7] Roughly speaking, in such an encryption scheme, given a ciphertext and a key, it is efficiently verifiable whether the given ciphertext was encrypted under the given key. This helps computing parties to correctly compute the garbled circuit.

- When input is 0, publish $E_y(1)$.
- When input is 1, publish a re-encryption of H (recall $H = (H_1, H_2) = E_y(h)$).

A proof of knowledge is added, $PK\{r : (c_1 = g^r,\ c_2 = y^r)$ or $(c_1 = H_1 g^r,\ c_2 = H_2 y^r)\}$.

Online Stage: Obtaining keys for input-wires. Computing parties need to obtain a key, for each wire j, that represents the Boolean value b that the corresponding input ciphertext encrypts – that is, w_b^j. But the key should not leak any information about the input ciphertext. Our protocol meets such requirement by using homomorphism of ElGamal encryption[8]. Let c_j be the ciphertext of contributed input $b \in \{0, 1\}$ for input wire j. Computing parties work as follows:

- For every input wire j, compute $W^j = W_0^j \cdot c_j$ locally using homomorphism of ElGamal encryption. Then, decrypt W^j via threshold decryption by computing parties using their shares for x. This gives w_b^j, which matches the input b.
- Each party computes the output of C using the key w_b^j locally.

Running the online stage in $\mathcal{P}_2$ takes one round. The communication complexity of $\mathcal{P}_2$ is $\mathcal{O}(n\mathsf{k}|C|\log|C|)$ (plus the NIZK, if we assume the CRS case) where $|C|$ is the size of the circuit.

3.3 Discussion

MP-CED vs. MPC with Preprocessing. General MPC and MP-CED can be reduced to each other.

- Given a protocol π for MP-CED, we can construct a protocol π' for MPC with preprocessing, as follows. In the preprocessing stage of π', the parties share an encryption key. In the online stage of π', each party publishes encryption of its input under the shared key, and the parties follow protocol π. The resulting MPC protocol π' requires one more online rounds than the underlying protocol π. This approach is implicitly used in [FH96, JJ00, CDN01, DN03].
- Given a protocol π' for MPC, we can construct a protocol π for MP-CED, as follows. In MP-CED, the parties share a secret key, and the inputs are encrypted. Protocol π should compute C on these given input ciphertexts. This can be done by the parties running protocol π' using a circuit C' derived from C. Circuit C' consists of two stages: the first stage of C' gets shares of the secret key and the ciphertexts, and decrypts the ciphertexts to give plaintexts. The second stage of C' essentially evaluates C on these plaintext inputs from the first stage. In running the protocol π, each party's input is its share for the secret key. Circuit C' has more gates than C. However, if the round complexity of π' does not depend on the depth of the circuit, then the round complexity of π is the same as the round complexity of π'.

On Basing MP-CED on Doubly-Homomorphic Encryption. Recently, Gentry constructed a doubly homomorphic encryption scheme using ideal lattices [G09], which

[8] In fact, any homomorphic encryption can be used. We chose to use ElGamal encryption since it is already used in $\mathcal{P}_1$.

solves the CED problem. Since our goal is to give a round-efficient protocol, it is an interesting question whether doubly-homomorphic encryption allows non-interactive secure computation. However, this seems unlikely.

- (Threshold Decryption.) It's not known whether Gentry's scheme supports threshold decryption. Thus, there has to be at least one party which can decrypt ciphertexts by itself. If this party sees the inputs (which are encrypted and published in the MP-CED model), it can decrypt private inputs of other parties and break the security. Thus, there must be a separation between parties who can decrypt and parties who get access to the input and intermediate ciphertexts.
- (Malicious Parties.) Parties without decryption capability would compute a circuit on encrypted inputs using double homomorphism. In order for the protocol to compute output in a plaintext form, they have to submit some ciphertexts to a party with decryption capability. In the malicious setting, to make sure that they applied doubly homomorphism correctly, some kind of zero-knowledge proof should be added to the ciphertexts they submit. However, it is not clear how such a proof can be constructed when the verifier has the decryption capability – as mentioned above, it must not see the input ciphertexts.

The above issue also stands against achieving MPC protocols against an active adversary with doubly homomorphic encryptions.

References

[AIK05] Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. In: IEEE Conference on Computational Complexity, pp. 260–274 (2005)

[B00] Beaver, D.: Minimal-latency secure function evaluation. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 335–350. Springer, Heidelberg (2000)

[BL96] Boneh, D., Lipton, R.: Algorithms for black-box fields and their application to cryptography. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 283–297. Springer, Heidelberg (1996)

[BMR90] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC), pp. 503–513 (1990)

[C01] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 136–145 (2001)

[CDN01] Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–299. Springer, Heidelberg (2001)

[CDS94] Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)

[CEJ$^+$07] Choi, S.G., Elbaz, A., Juels, A., Malkin, T., Yung, M.: Two-party computing with encrypted data. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 298–314. Springer, Heidelberg (2007)

[CF01] Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)

[CKL03] Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 68–86. Springer, Heidelberg (2003)

[CLOS02] Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: Proc. 34th Annual ACM Symposium on Theory of Computing (STOC), pp. 494–503 (2002)

[DH76] Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. on Information Theory IT-22(6), 644–654 (1976)

[DI05] Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (2005)

[DI06] Damgård, I., Ishai, Y.: Scalable secure multiparty computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 501–520. Springer, Heidelberg (2006)

[DIK+08] Damgård, I., Ishai, Y., Krøigaard, M., Nielsen, J.B., Smith, A.: Scalable multiparty computation with nearly optimal work and resilience. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 241–261. Springer, Heidelberg (2008)

[DN03] Damgård, I., Nielsen, J.B.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Heidelberg (2003)

[E85] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31, 469–472 (1985)

[F87] Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: Proc. 28th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 427–437 (1987)

[FH96] Franklin, M.K., Haber, S.: Joint encryption and message-efficient secure computation. joc. 9(4), 217–232 (1996)

[FS86] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Massey, J.L. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)

[G09] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC (to appear, 2009)

[GIKR01] Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: The round complexity of verifiable secret sharing and secure multicast. In: Proc. 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 580–589 (2001)

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proc. 19th Annual ACM Symposium on Theory of Computing (STOC), pp. 218–229. ACM Press, New York (1987)

[HK07] Horvitz, O., Katz, J.: Universally-composable two-party computation in two rounds. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 111–129. Springer, Heidelberg (2007)

[IK00] Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: FOCS, pp. 294–304 (2000)

[IKLP06] Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: On combining privacy with guaranteed output delivery in secure multiparty computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 483–500. Springer, Heidelberg (2006)

[IPS08] Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)

[JJ00] Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, p. 162. Springer, Heidelberg (2000)

[KO04] Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 335–354. Springer, Heidelberg (2004)

[KOS03] Katz, J., Ostrovsky, R., Smith, A.: Round efficiency of multi-party computation with a dishonest majority. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 578–595. Springer, Heidelberg (2003)

[KS08] Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008)

[LP09] Lindell, Y., Pinkas, B.: A proof of security of yao's protocol for two-party computation. J. Cryptology 22(2), 161–188 (2009)

[NO09] Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (2009)

[RAD78] Rivest, R., Adelman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: DeMillo, R., Dobkin, D., Jones, A., Lipton, R. (eds.) Foundations of Secure Computation, pp. 169–179. Academic Press, London (1978)

[S79] Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)

[SCO$^+$01] Santis, A.D., Crescenzo, G.D., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (2001)

[SYY99] Sander, T., Young, A., Yung, M.: Non-interactive cryptocomputing for NC^1. In: Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 554–567 (1999)

[TY98] Tsiounis, Y., Yung, M.: On the security of ElGamal based encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (1998)

[V76] Valiant, L.G.: Universal circuits (preliminary report). In: Proc. 8th Annual ACM Symposium on Theory of Computing (STOC), pp. 196–203 (1976)

[Y82] Yao, A.: Protocols for secure computations (extended abstract). In: Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 160–164 (1982)

[Y86] Yao, A.: How to generate an exchange secrets. In: Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 162–167 (1986)

A Explicit Preprocessing of $\mathcal{P}_1$

During the preprocessing stage, the parties can generate polynomial number of garbled gates, which can later be used for evaluating circuits. Therefore it suffices to know a bound on the sizes of circuits to be evaluated later. We show how to generate such truth tables explicitly given y as a global public key.

Throughout, each bit b will be encrypted with plaintext g^b. Denote by $\langle m \rangle$ a simple ElGamal ciphertext (with randomness $r = 0$): $(1, m)$. For an ElGamal ciphertext c for a bit, its negation $\neg c$ is defined as $\langle g^1 \rangle / c$. For two ElGamal ciphertext $a = (a_1, a_2)$ and $b = (b_1, b_2)$, define $\mathsf{ZKe}_u(a, b)$ — the proof that b is a re-encryption a with public key u — as $PK\{r : b_1 = g^r a_1,\ b_2 = u^r a_2\}$. When public key is not specified, ZKe means ZKe_y. The construction details can be found in Appendix A.3.

A.1 Preliminaries: Joint Generation of Garbled Gates

We associate a gate with the truth table for it. The entries of the truth tables are encrypted Boolean values, and the rows of each truth table are permuted, such that only a threshold of the parties can (1) recover any plaintext and (2) learn the permutation of the rows.

Sampling a Random Encrypted Boolean Value. In this protocol, n parties perform an oblivious analogue of XORing their respective random bits in n rounds. In our case, semantic security of ElGamal and the soundness of the attached proof guarantee they cannot.

1. Each party P_i selects $a_i \in_R \{0,1\}$ and computes $\widehat{a_i} = E_y(g^{a_i})$, $\pi_i = \mathsf{ZKe}(\langle g^0 \rangle, \widehat{a_i}) \vee \mathsf{ZKe}(\langle g^1 \rangle, \widehat{a_i})$ and broadcasts $(\widehat{a_i}, \pi_i)$. Let $S = \{j : \pi_j \text{ is valid}\}$. Set $\widehat{a} \leftarrow \widehat{a_{min}}$ where $min = \min_{j \in S} j$.
2. For $j = 2, \ldots, |S|$:

   Let i be the j-th smallest element in S. P_i computes an encryption $\widehat{d_i}$ such that $\widehat{d_i} = (d_{i1}, d_{i2})$ is a re-encryption of $\widehat{a}$ if $a_i = 0$ or a re-encryption of $\neg\widehat{a}$ otherwise. Then P_i broadcasts $(\widehat{d_i}, \psi_i)$ where
   $$\psi_i = \Big(\mathsf{ZKe}(\langle g^0 \rangle, \widehat{a_i}) \wedge \mathsf{ZKe}(\widehat{a}, \widehat{d_i})\Big) \vee \Big(\mathsf{ZKe}(\langle g^1 \rangle, \widehat{a_i}) \wedge \mathsf{ZKe}(\neg\widehat{a}, \widehat{d_i})\Big).$$
   If ψ_i is valid, then each party sets $\widehat{a} \leftarrow \widehat{d_i}$.

As in computing xor, it is enough that one of the bits is random (or, in our case, that one party is honest) to guarantee a random output as long as corrupt parties can not have their bit choices depend on the bits of other parties. The invariant of the protocol is that at the end of each round the ciphertext $\widehat{a}$ encrypts exclusive-or of a_i's so far.

Generating a Garbled IDENTITY Gate. First, run the procedure of sampling a random encrypted Boolean value. Let the output of the procedure is $\widehat{a}$. The first row of an IDENTITY gate is $\widehat{a}$, and the second row is computed by negating the value of $\widehat{a}$.

In	Out
$\widehat{a}$	$\widehat{a}$
$\neg\widehat{a}$	$\neg\widehat{a}$

IDENTITY

Generating a Garbled NAND Gate.

1. Each party P_i selects $a_i, b_i \in_R \{0,1\}$ and computes $\widehat{a_i} = E_y(g^{a_i})$, $\widehat{b_i} = E_y(g^{b_i})$, $\pi_i = \mathsf{ZKe}(\langle g^0 \rangle, \widehat{a_i}) \vee \mathsf{ZKe}(\langle g^1 \rangle, \widehat{a_i})$, and $\phi_i = \mathsf{ZKe}(\langle g^0 \rangle, \widehat{b_i}) \vee \mathsf{ZKe}(\langle g^1 \rangle, \widehat{b_i})$, and broadcasts $(\widehat{a_i}, \widehat{b_i}, \pi_i, \phi_i)$.
2. Run the procedure of sampling random encrypted Boolean values with $\widehat{a_i}$'s. Let $\widehat{a}$ be the output of the procedure. Let $S = \{j : \pi_j \text{ and } \phi_j \text{ are valid}\}$. Set $\widehat{b} \leftarrow \langle g^0 \rangle$ and $\widehat{ab} \leftarrow \langle g^0 \rangle$.
3. For $j = 1, \ldots, |S|$: Let i be the j-th smallest element in S. P_i computes encryptions $\widehat{d_i}$ and $\widehat{e_i}$ such that

   - If $b_i = 0$, then $\widehat{d_i}$ is a re-encryption of $\widehat{b}$ and $\widehat{e_i}$ is a re-encryption of $\widehat{ab}$.
   - If $b_i = 1$, then $\widehat{d_i}$ is a re-encryption of $\neg\widehat{b}$ and $\widehat{e_i}$ is a re-encryption of $\widehat{a}/\widehat{ab}$. Then P_i broadcasts $\big(\widehat{d_i}, \widehat{e_i}, \psi_i\big)$ where $\psi_i = \psi_i^0 \vee \psi_i^1$ for $\psi_i^0 = \mathsf{ZKe}(\langle g^0 \rangle, \widehat{b_i}) \wedge \mathsf{ZKe}(\widehat{b}, \widehat{d_i}) \wedge \mathsf{ZKe}(\widehat{ab}, \widehat{e_i})$ and $\psi_i^0 = \mathsf{ZKe}(\langle g^1 \rangle, \widehat{b_i}) \wedge \mathsf{ZKe}(\neg\widehat{b}, \widehat{d_i}) \wedge \mathsf{ZKe}(\widehat{a}/\widehat{ab}, \widehat{e_i})$. If ψ_i is valid, then each party sets $\widehat{b} \leftarrow \widehat{d_i}$ and $\widehat{ab} \leftarrow \widehat{e_i}$.

The invariant of the loop is that at the end of each round the ciphertext $\widehat{ab}$ encrypts exclusive-or of ab_i's so far. After $\widehat{a}, \widehat{b}, \widehat{ab}$ are generated, each party P_i can complete the truth table, by locally negating the ciphertexts as described in the table.

In_L	In_R	Out
$\widehat{a}$	$\widehat{b}$	$\neg\widehat{ab}$
$\widehat{a}$	$\neg\widehat{b}$	$\widehat{ab} \cdot (\neg\widehat{a})$
$\neg\widehat{a}$	$\widehat{b}$	$\widehat{ab} \cdot (\neg\widehat{b})$
$\neg\widehat{a}$	$\neg\widehat{b}$	$\widehat{a} \cdot \widehat{b}/\widehat{ab}$

NAND

A.2 Preliminaries: Jointly Recoverable Encrypted ElGamal Key Pairs

Verifiable ElGamal Encryption of Discrete Logarithm. To generate a jointly recoverable encrypted ElGamal key pair, we first introduce the following verifiable encryption of discrete logarithm.

Let $\gamma := g^z$. We want to encrypt z in a verifiable manner. Let z_i be the i-th rightmost bit of z for $i \in [\mathsf{k}]$. The verifiable encryption is $\hat{E}_y(z) = \big(\widehat{z_0}, \ldots, \widehat{z_{\mathsf{k}-1}}, \pi\big)$, where $\widehat{z_i} = E_y(g^{z_i \cdot 2^i})$ for $i \in [\mathsf{k}]$. The proof π is

$$\Big(\bigwedge_{i=0}^{\mathsf{k}-1}\Big(\mathsf{ZKe}(\langle g^0\rangle, \widehat{z_i}) \vee \mathsf{ZKe}(\langle g^{2^i}\rangle, \widehat{z_i})\Big)\Big) \wedge \mathsf{ZKe}\Big(\langle\gamma\rangle, \prod_{i=0}^{\mathsf{k}-1} \widehat{z_i}\Big).$$

When we get $(g^{z_0 \cdot 2^0}, \ldots, g^{z_{\mathsf{k}-1} \cdot 2^{\mathsf{k}-1}})$ by decrypting $\hat{E}_y(z)$, z can be extracted via exhaustive search in polynomial time in k because each z_i is a bit.

Note that the encryption scheme is homomorphic if we ignore the proof part. Multiplication of two verifiable encryptions $\hat{E}_y(z) = (\widehat{z_1}, \ldots, \widehat{z_{\mathsf{k}-1}})$ and $\hat{E}_y(w) = (\widehat{w_1}, \ldots, \widehat{w_{\mathsf{k}-1}})$ is defined as $\hat{E}_y(z) \cdot \hat{E}_y(w) = (\widehat{z_1} \cdot \widehat{w_1}, \ldots, \widehat{z_{\mathsf{k}-1}} \cdot \widehat{w_{\mathsf{k}-1}})$.

Generation of Jointly Recoverable Encrypted ElGamal Key Pairs. For simplicity, we omit the proof part of the verifiable encryption from the presentation below. Generation of a key pair can be done as follows:

1. Each party P_j runs ElGamal key generation and obtains (k_j, g^{k_j}). It broadcasts $(g^{k_j}, \hat{E}_y(k_j))$.
2. Let S be the set of parties whose encryptions are verified. In the PK column, $\prod_{j \in S} g^{k_j}$ is set. In the SK column, $\prod_{j \in S} \hat{E}_y(k_j)$ is set.

Extraction of the secret key. Let $(Y_0, \ldots, Y_{\mathsf{k}-1}) := \prod_{j \in S} \hat{E}_y(k_j)$. Let g^{z_i} be the decryption of Y_i. Then given $(g^{z_0}, \ldots, g^{z_{\mathsf{k}-1}})$, we can extract the secret key $\sum_{j \in S} k_j = \sum_i z_i$ by finding each z_i via exhaustive search, which can be done efficiently since $g^{z_i} \in \{2^{0 \cdot 2^i}, 2^{1 \cdot 2^i}, \ldots, 2^{n \cdot 2^i}\}$.

A.3 Preprocessing of $\mathcal{P}_1$

The preprocessing takes $2n$ rounds, since step 1.1 and step 1.2 can be executed concurrently. This protocol is UC-secure, but for lack of space, we defer the proof of security to full version.

Step 1.1: Garbled Circuit Generation - Intermediate Gates. For each NAND gate, run the procedure of joint generation of garbled NAND gate in Appendix A.1 to fill in In and Out Columns. For each pair of columns PK and SK, run the procedure of jointly recoverable encrypted ElGamal key pairs in Appendix A.2.[9] The above tasks are executed in parallel.

Step 1.2: Garbled Circuit Generation - Output Gates

1. Run the procedure of sampling random encrypted Boolean values in Appendix A.1 where each party P_i selects $a_i \in_R \{0, 1\}$. Let $\widehat{a}$ be the output of the procedure and let $S = \{j : P_j \text{ behaved honestly during the procedure}\}$. Fill in In and Out Columns as an IDENTITY gate.

[9] Now in the online stage, k instances of M-CODE are executed since $T_b[\mathsf{SK}][w]$ contains k ElGamal ciphertexts. The communication complexity blows up by multiplicative factor of k.

In addition, run the procedures of jointly recoverable encrypted ElGamal key pairs in Appendix A.2 to fill the columns PK and SK. Let z_1 and z_2 be the two keys in the column PK.

2. In order to fill Final column, each party P_i such that $i \in S$ broadcasts $(\widehat{a_{i,z_1}} = E_{z_1}(g^{a_i}), \widehat{a_{i,z_2}} = E_{z_2}(g^{a_i}))$. Set $\widehat{a_{z_1}} \leftarrow \langle g^0 \rangle$, $\widehat{a_{z_2}} \leftarrow \langle g^1 \rangle$. Parties jointly compute $E_{z_1}(g^{\bigoplus_i a_i})$ and $E_{z_2}(g^{1-\bigoplus_i a_i})$. In particular, for $i = 1, \ldots, |S|$:

   (a) Let i be the j-th smallest element in S. P_i computes encryptions $\widehat{d_i}$, $\widehat{e_i}$ such that $\widehat{d_i}$ (resp. $\widehat{e_i}$) is a re-encryption of $\widehat{a_{i,z_1}}$ (resp. $\widehat{a_{i,z_2}}$) if $a_i = 0$ or a re-encryption of $\neg\widehat{a_{i,z_1}}$ (resp. $\neg\widehat{a_{i,z_2}}$) otherwise. Then P_i broadcasts $\left(\widehat{d_i}, \widehat{e_i}, \psi_i\right)$ where $\psi_i = \psi_i^0 \vee \psi_i^1$ for

   $$\begin{aligned} \psi^b e &= \mathsf{ZKe}(\langle g^0 \rangle, \widehat{a_i}) \wedge \mathsf{ZKe}_{z_1}(\langle g^0 \rangle, \widehat{a_{i,z_1}}) \wedge \mathsf{ZKe}_{z_2}(\langle g^0 \rangle, \widehat{a_{i,z_2}}) \wedge \\ &\quad \mathsf{ZKe}_{z_1}(\widehat{a_{z_1}}, \widehat{d_i}) \wedge \mathsf{ZKe}_{z_2}(\widehat{a_{z_2}}, \widehat{e_i}) \quad \text{and} \\ \psi_i^1 &= \mathsf{ZKe}(\langle g^1 \rangle, \widehat{a_i}) \wedge \mathsf{ZKe}_{z_1}(\langle g^1 \rangle, \widehat{a_{i,z_1}}) \wedge \mathsf{ZKe}_{z_2}(\langle g^1 \rangle, \widehat{a_{i,z_2}}) \wedge \\ &\quad \mathsf{ZKe}_{z_1}(\neg\widehat{a_{z_1}}, \widehat{d_i}) \wedge \mathsf{ZKe}_{z_2}(\neg\widehat{a_{z_2}}, \widehat{e_i}). \end{aligned}$$

   (b) If ψ_i is valid, then each party sets $\widehat{a_{z_1}} \leftarrow \widehat{d_i}$, and $\widehat{a_{z_2}} \leftarrow \widehat{e_i}$. Otherwise, in the case of honest majority, parties collectively compute a_i from threshold decryption using $(y_1, \ldots, y_n)$ and compute $\widehat{a_{z_1}}, \widehat{a_{z_2}}$ accordingly. In the case of honest minority, the protocol aborts. Finally, set $\widehat{a_{z_2}} \leftarrow \neg\widehat{a_{z_2}}$.