Secure Multiparty Computation of DNF

Kun Peng

Institute for Infocomm Research

Abstract. Homomorphism based multiparty computation techniques are studied in this paper as they have several advantages over the other multiparty computation schemes. A new homomorphism based multiparty computation technique is proposed to evaluate functions in DNF form. The new technique exploits homomorphism of a certain sealing function to evaluate a function in DNF. The new technique has two advantages over the existing homomorphism based multiparty computation schemes. Firstly, it supports any input format. Secondly, a general method to reduce any function to DNFs is proposed in this paper. With this method, functions like the famous millionaire problem can be reduced to DNFs and efficiently evaluated. Security of the new scheme is formally defined in the static active adversary model and proved in a new simulation model.

1 Introduction

Secure multiparty computation [1, 2, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 23, 24, 25, 26, 27, 28] is a technique to evaluate a function without revealing any information about the inputs except the output. The basic technique of multiparty computation is to present the function in a circuit composed of a few logic gates and reduce the computation to evaluation of each gate. The inputs to the function must be sealed and output of each gate must be private so that privacy of the multiparty computation is protected.

Most existing multiparty computation schemes garble the inputs and outputs wires of the gates to achieve privacy and this mechanism has a few drawbacks. Firstly, they do not provide any concrete method to design an evaluation circuit for the evaluated function. Instead they usually assume that the circuit already exists and is ready to be garbled. Without a concrete circuit, the consequent operations and analysis are based on assumptions like "after the circuit is generated by a party..." or "if the number of levels of gates is a logarithm of the length of all the inputs ...". Secondly, they need either a single circuit generator (who knows how each gate is garbled and thus can learn additional information about the inputs by monitoring execution of function evaluation) or a complex and inefficient distributed multiparty circuit generation algorithm. Thirdly, it is complex and inefficient for them to publicly prove and verify correctness of the circuit without compromising its privacy. Fourthly, it is complex and inefficient to match all the input variables to the function with the garbled inputting wires

S. Qing, H. Imai, and G. Wang (Eds.): ICICS 2007, LNCS 4861, pp. 254–268, 2007.

of the circuit, especially when public verifiability of this matching is required. Finally, efficiency is low when appropriate trust sharing, public verifiability and provable security are required.

We are interested in another solution [1, 9, 11, 12, 16, 23, 24, 25, 26, 28], which employs homomorphic sealing function to seal the inputs and exploit homomorphism of the sealing function to implement multiparty computation. Intermediate outputs of all the gate are sealed, so does not reveal any information assuming the sealing function is difficult to break. So the circuit can be public and no gate needs garbling. Therefore the drawbacks of the first solution are overcome. However, this solution has its own drawbacks. Firstly, only several certain kinds of gates are supported and thus the function must be in a special format. [9, 11] require that the function only employs three kinds of gates: "+", "-" and multiplication. [1, 28] only support NOT and OR gates, which [12] only supports XOR and AND gates. [16] requires a very special format for the function: Standard low-degree polynomials over a finite field. Obviously, given a random function it is possible that it cannot be reduced to a polynomial number of gates in a special format. So each scheme in the second solution has its own favourite functions, which can be reduced to a polynomial number of gates in the special format in that scheme and thus can be efficiently evaluated. Moreover, each of them requires that each input variable must be in a special format and they fail if any input variable is invalid. In addition, some schemes [23, 24, 25, 26] are only suitabe to handle some certain functions.

In this paper, a new homomorphism based secure multiparty computation scheme is proposed. Any function in DNF (disjunctive normal form) form[1] can be efficiently evaluated in the new scheme. It has two main advantages over the existing homomorphism based multiparty computation schemes. Firstly, it does not require any special input format. Instead, any input format is supported and an appropriate format can be flexibly chosen for the evaluated function. Secondly, a general method to reduce any function to DNFs is proposed in this paper. With this method, functions like the famous millionaire problem can be reduced to simple DNFs and then efficiently evaluated. The new scheme employs a flexible participant model. Like [10], it does not require all the input providers to take part in the computation. Although there may be many inputs and input providers to a function, a small number of computation performers can be employed, such that communication between them is not a heavy overhead in practice. This participant model is especially suitable for applications like e-auction and e-voting. Static active adversary model is used in the new scheme, which is strong enough for most practical applications. The UC (universal composable) security model [4] is not employed in the new scheme. Correctness and soundness of the new scheme are defined in a straightforward manner while its privacy is defined in a new simulation model detailed in Section 2, which is simpler and more practical than the UC model and other traditional simulation models. Security of the new scheme is formally guaranteed when a majority of the computation performers are honest.

[1] Detailed definition of function in DNF will be given in Section 4.

2 Security Model

There are usually three kinds of participants in multiparty computation: IPs (input providers), CPs (computation performers) and result receivers. They respectively provide inputs, carry out the computation and receive the result. Although it is assumed in most multiparty computation schemes that all the participants play all the three roles, that assumption is impractical in many applications like e-auction and e-voting. A typical application is inputs from many IPs are evaluated by several CPs. So in this paper, the participant model in [10] is adopted: the IPs and the CPs are not the same group of participants and the number of CPs is small. However, our participant model is a general model and the three sets of players need not be disjoint. In garbled circuit based secure computation, more participants like circuit generator (called compiler or issuer in some schemes) are often needed. In this paper, we design a homomorphism based secure computation scheme, so only need the three kinds of participants. When we say a CP is corrupted, we mean an adversary obtains its complete secret information (including historical record) and controls it behaviour. In other words, we adopt active adversary model. Moreover, we assume the adversaries are static. Two kinds of synchronous communication channels are used in this paper. A public broadcast channel also called bulletin board is set up for exchange of public information. In addition, there is an authenticated confidential channel from each IP to each CP. The following security properties are required in a multiparty computation protocol.

- Correctness: when given encryption (or commitment) of inputs $x_1, x_2, \ldots, x_n$ and asked to compute function $f()$, if a majority of CPs are not corrupted, the protocol outputs $f(x_1, x_2, \ldots, x_n)$.
- Public verifiability: there is a public verification procedure, by which any one can publicly check whether the protocol outputs $f(x_1, x_2, \ldots, x_n)$ when inputs $x_1, x_2, \ldots, x_n$ are given to function $f()$.
- Soundness: if a majority of CPs pass the public verification procedure, the protocol outputs $f(x_1, x_2, \ldots, x_n)$ when given encryption (or commitment) of input $x_1, x_2, \ldots, x_n$ and $f()$.
- Privacy: if a majority of CPs are not corrupted no information about the input is revealed except what can be deduced from the result of the function.

In the security definition above, the complex UC model [4] is not used. Correctness and soundness are more straightforward defined. However, definition of privacy is still intuitive and informal. So it is more formally defined in a simulation model simpler than the UC model as follows.

Definition 1. *There exists a polynomial algorithm for a party without any knowledge about any input to simulate the transcript of the secure computation protocol such that no polynomial algorithm can distinguish the simulated transcript and the real transcript with a probability non-negligibly larger than 0.5.*

3 Parameters and Primitives

p, q, G, g, h are public parameters. p is a large prime such that $p-1$ has a large factor q with no small factor. G is the cyclic subgroup of Z_p^* with order q. g and h are generators of G such that $\log_g h$ is unknown. A primitive to be used later, the t-out-of-m secret sharing algorithm in [22], is described as follows.

- To share a secret s among m parties $A_1, A_2, \ldots, A_m$, a dealer builds two polynomials $F(x) = \sum_{l=0}^{t-1} f_l x^l \bmod q$ and $H(x) = \sum_{l=0}^{t-1} h_l x^l \bmod q$ where $f_0 = s$ and f_l for $l = 1, 2, \ldots, t-1$ and h_l for $l = 0, 1, \ldots, t-1$ are random integers in Z_q.
- The dealer publishes sharing commitments $E_l = g^{f_l} h^{h_l} \bmod p$ for $l = 0, 1, \ldots, t-1$ on the bulletin board.
- The dealer sends A_k its share $(s_k, r_k) = (F(k), H(k))$ through the authenticated confidential channel.
- A_k verifies $g^{s_k} h^{r_k} = \prod_{l=0}^{t-1} E_l^{k^l} \bmod p$. If the verification is passed, A_k can be sure that (s_k, r_k) is the k^{th} share of the secret committed in E_0. As m and t are usually small integers, each verification costs 2 exponentiations and $O(t)$ multiplications.
- If at least t correct shares are put together, the secret committed in E_0 can be recovered as $s = \sum_{k \in \Phi} s_k u_k \bmod q$ where $u_k = \prod_{l \in \Phi, l \neq k} \frac{l}{l-k}$ and Φ is a set containing the indexes of t correct shares.

Note that definition of u_k will be used throughout the paper. Pedersen [22] proves that when $\log_g h$ is unknown and discrete logarithm is a hard problem, there is only one polynomial way to open commitment E_0, which is denoted as $s \longleftarrow REC(E_0)$ in this paper. Pedersen also illustrates that his secret sharing scheme is homomorphic. Namely, $REC(E_0) + REC(E_0') = REC(E_0 E_0') \bmod q$ where E_0 and E_0' are the first components in two commitments.

Besides the general secret sharing algorithm described above, a special variant of it is employed in this paper. In the special variant, a public known integer s instead of a secret is shared. Its purpose is not to secretly hide the integer but to publicly distribute it into a sharing format. Implementation of the special secret sharing algorithm is simple: the commitment generation and sharing generation function are the same as in the general secret sharing algorithm except that $F(x) = s$, $H(x) = 0$ and the shares are public. So anyone can calculate commitment $g^s, 1, 1, \ldots, 1$ and each A_k's share is $(s, 0)$. This special variant is called simplified secret sharing, which only costs one exponentiation but has all the properties of the original secret sharing algorithm except privacy of the shared integer.

In this paper, ElGamal encryption is employed. Private key x_k is chosen for A_k from Z_q. A_k's public key y_k is $g^{x_k} \bmod p$. Note that ElGamal encryption is semantically secure. More precisely, given a ciphertext c and two messages m_1 and m_2, such that $c = E(m_i)$ where $i = 1$ or 2, there is no polynomial algorithm to find out i with a probability non-negligibly larger than 0.5 when the private key is unknown. In this paper, $ZP\ [\ x_1, x_2, \ldots, x_\alpha \mid R_1, R_2, \ldots, R_\beta\ ]$ denotes a ZK proof of knowledge of $x_1, x_2, \ldots, x_\alpha$ satisfying conditions $R_1, R_2, \ldots, R_\beta$. In

this paper, there is a security parameter T such that T is a small integer and 2^T is no larger than the smallest factor of q.

4 Reduction to DNFs and Sealing the Inputs

A function with μ IPs $IP_1, IP_2, \ldots, IP_\mu$ and inputs $m_1, m_2, \ldots, m_\mu$ can be reduced to DNFs as follows where m_ι is the input of IP_ι for $\iota = 1, 2, \ldots, \mu$.

1. Integer L is chosen such that $2^L \geq S$ and $2^{L-1} < S$ where S is the size of the co-domain of the function. Sort all the variables in the co-domain in a certain order and the k^{th} variable is transformed into a new format: the binary representation of k.
2. The function is divided into L sub-functions, each of which receives the input of the function and outputs one bit of the result of the function in the new output format.
3. Each sub-function is transformed into a DNF as follows.
   (a) The truth table mapping $m_1, m_2, \ldots, m_\mu$ through the sub-function to $\{0, 1\}$ is established.
   (b) Every row with an output 1 in the truth table is picked out. Each chosen row is transformed into a clause, which tests whether m_ι equals to the ι^{th} input variable in the row for $\iota = 1, 2, \ldots, \mu$. Each test in the clause is a basic logic computations and they are linked with AND logic.
   (c) Linking all the clauses with OR logic produces a DNF.
   (d) Two methods can be employed to optimisation DNFs. The first method adjusts the length and number of the inputs. A long input is divided into multiple shorter inputs such that the the number of rows in the truth table decreases. The second method is Karnaugh map, which simplifies a DNF into least minterm form (See Pages 104–106 of [20]). Both methods are tried until the simplest DNFs are obtained after multiple trials.
4. Outputs of all the DNFs form a L bit binary string, which can be transformed back into the functions' original co-domain (the bit string representing integer k is transformed to the k^{th} output in the original co-domain).

Note that the algorithm above is not unique. For example, Karnaugh map can be replaced by Quine-McCluskey technique (See Page 99 of [20]). After the simplest DNF circuit is obtained, the function can be efficiently evaluated as described later in this paper. An example will be given in Section 7 to illustrate high efficiency of this method.

To protect privacy of secure multiparty computation, the inputs of any DNF must be sealed when it is processed. The unsealing power is shared among the CPs.In our solution, the secret sharing algorithm in Section 3 is employed to seal the inputs where the CPs $A_1, A_2, \ldots, A_m$ act as the share holders. The inputs to the DNFs of the function are sealed as follows using the t-out-of-m verifiable secret sharing scheme in [22] as described in Section 3 and we require $m+1 = 2t$.

1. For $\iota = 1, 2, \ldots, \mu$ each IP_ι seals m_ι in commitment $E_{\iota,l}$ for $l = 0, 1, \ldots, t-1$ and shares it among the CPs. The commitments are published on the

bulletin board and share $(s_{\iota,k}, r_{\iota,k})$ is sent to A_k through the authenticated confidential channel for $k = 1, 2, \ldots, m$. Thus all the inputs to all the DNFs are sealed.

2. Every A_k verifies validity of each of his shares: $g^{s_{\iota,k}} h^{r_{\iota,k}} \stackrel{?}{=} \prod_{l=0}^{t-1} E_{\iota,l}^{k^l} \bmod p$ for $\iota = 1, 2, \ldots, \mu$. If an invalid share is found by a CP, it is published on the bulletin board. As the communication channel used to distribute the shares is authenticated, no IP can deny any invalid share it sends out. Therefore, dishonest IPs can be detected and expelled.
3. All the DNFs are transformed into sealed format (composed of sealed inputs and logic operation on them) one by one. A DNF in the form of (2) is transformed to

$$\vee_{i=1}^{n} \left(REC(E_{i,1,0}) = a_{i,1} \wedge REC(E_{i,2,0}) = a_{i,2} \wedge \ldots \wedge REC(E_{i,M(i),0}) = a_{i,M(i)}\right) \tag{1}$$

where $E_{i,j,l} = E_{\iota,l}$ for $l = 0, 1, \ldots, t-1$ if $m_{i,j} = m_\iota$. For $k = 1, 2, \ldots, m$ each A_k holds $(s_{\iota,k}, r_{\iota,k})$ as his share of $REC(E_{i,j,0})$ if $m_{i,j} = m_\iota$.

In each DNF, only the inputs to the DNF are sealed while all the equations, all the AND and OR logic relations and the way they are combined and linked are public. So without any proof anyone can publicly and directly check that in our new scheme that each DNF is correctly organised and all the DNFs cooperate to evaluate the target function.

5 Evaluation of DNF

A DNF is in the form of

$$\vee_{i=1}^{n} \ (m_{i,1} = a_{i,1} \wedge m_{i,2} = a_{i,2} \wedge \ldots \wedge m_{i,M(i)} = a_{i,M(i)}) \tag{2}$$

where $M(i)$ is the number of inputs involved in the i^{th} clause in that DNF and $m_{i,j} \in \{m_1, m_2, \ldots, m_\mu\}$. In DNF (1) there are three levels of computation. The bottom level is the equations; the middle level is AND logic and the top level is OR logic. The three levels are computed one by one from the bottom to the top.

5.1 Computation on the Bottom Level

By simply exploiting homomorphism of the employed secret sharing algorithm, (1) is simplified to

$$\begin{aligned} \vee_{i=1}^{n} \ (REC(g^{a_{i,1}}/E_{i,1,0}) = 0 \ \wedge \ REC(g^{a_{i,2}}/E_{i,2,0}) = 0 \ \wedge \\ \ldots \wedge \ REC(g^{a_{i,M(i)}}/E_{i,M(i),0}) = 0) \end{aligned} \tag{3}$$

Every commitment variable and share in the simplified DNF must be adjusted as follows.

1. A set $A = \bigcup_{i=1,j=1}^{i<n,j<M(i)} \{a_{i,j}\}$ is set up to include all the constant integers involved in all the equations in the DNF in the form (1). Suppose A has a size ν and $A = \{a_1, a_2, \ldots, a_\nu\}$.

2. For $\iota = 1, 2, \ldots, \nu$, commitment $\hat{E}_{\iota,l}$ for $l = 0, 1, \ldots, t-1$ and shares $(\hat{s}_{\iota,k}, \hat{r}_{\iota,k})$ for $k = 1, 2, \ldots, m$ are publicly available for a_ι using the simplified secret sharing algorithm in Section 3.
3. For each $a_{i,j}$, if $a_{i,j} = a_\iota$, then $\hat{E}_{i,j,l} = \hat{E}_{\iota,l}$ for $l = 0, 1, \ldots, t-1$ and $\hat{s}_{i,j,k} = \hat{s}_{\iota,k}$, $\hat{r}_{i,j,k} = \hat{r}_{\iota,k}$ for $k = 1, 2, \ldots, m$.
4. (3) is presented in the form

$$\vee_{i=1}^{n} \ (REC(E'_{i,1,0}) = 0 \wedge REC(E'_{i,2,0}) = 0 \wedge \ldots \wedge REC(E'_{i,M(i),0}) = 0) \quad (4)$$

where every commitment variable for (4), $E'_{i,j,l}$, is publicly available as $\hat{E}_{i,j,l}/E_{i,j,l} \bmod p$ for $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, M(i)$ and $l = 0, 1, \ldots, t-1$. A_k calculates $s'_{i,j,k} = \hat{s}_{i,j,k} - s_{i,j,k} \bmod q$ and $r'_{i,j,k} = \hat{r}_{i,j,k} - r_{i,j,k} \bmod q$ as its share of $REC(E'_{i,j,0})$ for $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, M(i)$ and $k = 0, 1, \ldots, m$.

5.2 Computation on the Middle Level

Homomorphism of the employed secret sharing algorithm is further exploited to transform (4) into

$$\vee_{i=1}^{n} \ \sum_{j=1}^{M(i)} REC(E'_{i,j,0}) t_j = 0 \quad (5)$$

where each t_j is a random integer.

1. The CPs cooperate to choose T bit random integers t_j for $j = 1, 2, \ldots, M$ where $M = max(M(1), M(2), \ldots, M(n))$. More precisely, each A_k secretly chooses random integers $t_{j,k}$ from Z_q for $j = 1, 2, \ldots, M$ and publishes $h_{j,k} = h(t_{j,k})$ for $j = 1, 2, \ldots, M$ where $h()$ is a collision resistant one-way function. After each $h_{j,k}$ has been published, the CPs publish $t_{j,k}$ for $j = 1, 2, \ldots, M$ and $k = 1, 2, \ldots, m$. Finally, $h_{j,k} = h(t_{j,k})$ for $j = 1, 2, \ldots, M$ and $k = 1, 2, \ldots, m$ is verified and $t_j = \sum_{k=1}^{m} t_{j,k} \bmod 2^T$ for $j = 1, 2, \ldots, M$ are calculated. For the sake of high efficiency, $h()$ can be a hash function. If there is any concern for collision resistance in hash functions, $h(x)$ can be $g^x \bmod p$.
2. (5) is presented in the form of (6).

$$\vee_{i=1}^{n} \ REC(E'_{i,0}) = 0 \quad (6)$$

where every commitment variable for (6), $E'_{i,l}$, is publicly available as $\prod_{j=1}^{M(i)} {E'}_{i,j,l}^{t_j} \bmod p$ for $i = 1, 2, \ldots, n$ and $l = 0, 1, \ldots, t-1$. A_k calculates $s'_{i,k} = \sum_{j=1}^{M(i)} s'_{i,j,k} t_j \bmod q$ and $r'_{j,k} = \sum_{j=1}^{M(i)} r'_{i,j,k} t_j \bmod q$ as its share of $REC(E'_{i,0})$ for $i = 1, 2, \ldots, n$ and $k = 0, 1, \ldots, m$.

5.3 Computation on the Top Level

(6) is equivalent to

$$\prod_{i=1}^{n} REC(E'_{i,0}) = 0,$$

which according to homomorphism of the employed secret sharing scheme is equivalent to

$$REC(E'^{\prod_{i=2}^{n} REC(E'_{i,0})}_{1,0}) = 0 \qquad \text{or namely} \qquad REC(E'^{\prod_{i=2}^{n} \sum_{k\in K} s_{i,k}u_k}_{1,0}) = 0,$$

which is equivalent to

$$REC((\prod_{k\in K} E'^{s'_{2,k}u_k}_{1,0})^{\prod_{i=3}^{n} \sum_{k\in K} s'_{i,k}u_k}) = 0, \tag{7}$$

where K is a set containing the indices of t honest CPs and $u_k = \prod_{l\in K, l\neq k} \frac{l}{l-k}$.

$\prod_{k\in K} E'^{s'_{2,k}u_k}_{1,0}$ is denoted as $E''_{1,0}$. To evaluate (7), $E''_{1,0}$ has to be calculated without revealing $s'_{2,k}$ for $k \in K$. Moreover, $E''_{1,l}$ for $l = 1, 2, \ldots, t-1$ have to be calculated to form a complete commitment. In addition, each A_k with k in K should get its share necessary to reconstruct the secret in $E''_{1,0}$. The commitment generation and shares distribution operations are as follows.

1. For $k \in K$ each A_k calculates and publishes $c'_{1,k} = (g^{\gamma_{1,k}} \bmod p,\ y_k^{\gamma_{1,k}} g^{s'_{1,k}} \bmod p)$ and $e'_{1,k} = (g^{\delta_{1,k}} \bmod p,\ y_k^{\delta_{1,k}} h^{r'_{1,k}} \bmod p)$ where $\gamma_{1,k}$ and $\delta_{1,k}$ are randomly chosen from Z_q.
2. For $k' \in K$ each $A_{k'}$ calculates $E''_{1,l,k'} = E'^{s'_{2,k'}u_{k'}}_{1,l} \bmod p$ for $l = 0, 1, \ldots, t-1$ and $c''_{1,k,k'} = c'^{s'_{2,k'}u_{k'}}_{1,k} \bmod p$, $e''_{1,k,k'} = e'^{s'_{2,k'}u_{k'}}_{1,k} \bmod p$ for $k \in K$.
3. $E''_{1,l} = \prod_{k'\in K} E''_{1,l,k'} \bmod p$ for $l = 0, 1, \ldots, t-1$ and $c''_{1,k} = \prod_{k'\in K} c''_{1,k,k'} \bmod p$, $e''_{1,k} = \prod_{k'\in K} e''_{1,k,k'} \bmod p$ for $k \in K$.

Note that in the algorithm above, two integers k' and k are used for the indices of the CPs. The reason is that each CP have two roles: share holder and evaluator. So two integers are needed for each index: A_k stands for the k^{th} CP holding $(s'_{1,k}, r'_{1,k})$ while $A_{k'}$ stands for the k'^{th} CP raising $(c'_{1,k}, e'_{1,k})$ and the commitment variables to the power of its secret $s'_{2,k'}u_{k'}$. Thus $E''_{1,0}$ and its subsidiary commitment variables are generated and each honest CP gets a encrypted share necessary to reconstruct the secret in $E''_{1,0}$. Therefore, (7) is transformed to

$$REC(E''^{\prod_{i=3}^{n} \sum_{k\in K} s'_{i,k}u_k}_{1,0}) = 0 \tag{8}$$

with the corresponding commitment variables and shares publicly available. (8) is equivalent to

$$REC((\prod_{k\in K} E''^{s'_{3,k}u_k}_{1,0})^{\prod_{i=4}^{n} \sum_{k\in K} s'_{i,k}u_k}) = 0 \tag{9}$$

(9) is then transformed to

$$REC((\prod_{k\in K} E''^{s'_{4,k}u_k}_{2,0})^{\prod_{i=5}^{n} \sum_{k\in K} s'_{i,k}u_k}) = 0, \tag{10}$$

where $E''_{2,l} = \prod_{k'\in K} E''^{s'_{3,k'}u_{k'}}_{1,l} \bmod p$ for $l = 0, 1, \ldots, t-1$ and the corresponding encrypted shares are $c''_{2,k} = \prod_{k'\in K} c''^{s'_{3,k'}u_{k'}}_{1,k} \bmod p$, $e''_{2,k} = \prod_{k'\in K} e'^{s'_{3,k'}u_{k'}}_{1,k}$

mod p for $k \in K$. (9) is transformed to (10) in the same way as (7) is transformed to (8). More precisely, each $A_{k'}$ with $k' \in K$ raises the commitment variables and encrypted shares of (9) to the power of his secret $s_{3,k'}u_{k'}$ and then the honest CPs' outputs are combined.

The transform continues until (6) is reduced to

$$REC(\prod_{k\in K} {E''}_{n-2,0}^{s'_{n,k}u_k}) = 0$$

(with supporting commitment variables ${E''}_{n-2,l}$ for $l = 1, 2, \ldots, t-1$ and corresponding encrypted shares $c''_{n-2,k}$, $e''_{n-2,k}$ for $k \in K$) and finally

$$REC(E''_{n-1,0}) = 0 \tag{11}$$

with supporting commitment variables ${E''}_{n-1,l}$ for $l = 1, 2, \ldots, t-1$ and corresponding encrypted shares $c''_{n-1,k}$, $e''_{n-1,k}$ for $k \in K$ where $E''_{n-1,l} = \prod_{k'\in K} {E''}_{n-2,l}^{s'_{n,k'}u_{k'}} \bmod p$ and $c''_{n-1,k} = \prod_{k'\in K} {c''}_{n-2,k}^{s'_{n,k'}u_{k'}} \bmod p$, $e''_{n-1,k} = \prod_{k'\in K} {e'}_{n-2,k}^{s'_{n,k'}u_{k'}} \bmod p$ for $k \in K$.

All the operations described intuitively above in this subsection can be described in an abstract manner as follows. For $i = 2, 3, \ldots, n$:

1. for $k' \in K$ each $A_{k'}$ calculates

$$E''_{i-1,l,k'} = {E''}_{i-2,l}^{s'_{i,k'}u_{k'}} \bmod p \text{ for } l = 0, 1, \ldots, t-1 \tag{12}$$

$$c''_{i-1,k,k'} = {c''}_{i-2,k}^{s'_{i,k'}u_{k'}} \bmod p \text{ for } k \in K \tag{13}$$

$$e''_{i-1,k,k'} = {e''}_{i-2,k}^{s'_{i,k'}u_{k'}} \bmod p \text{ for } k \in K \tag{14}$$

2. $E''_{i-1,l} = \prod_{k'\in K} E''_{i-1,l,k'}$ for $l = 0, 1, \ldots, t-1$ and $c''_{i-1,k} = \prod_{k'\in K} c''_{i-1,k,k'}$, $e''_{i-1,k} = \prod_{k'\in K} e''_{i-1,k,k'}$ for $k \in K$

where $E''_{0,l} = E'_{1,l}$ for $l = 0, 1, \ldots, t-1$ and $c''_{0,k} = c'_{1,k}$ for $k \in K$, $e''_{0,k} = e'_{1,k}$ for $k \in K$.

5.4 Secret Reconstruction

(11) is solved as follows.

1. For $k \in K$ each A_k decrypts his encrypted share $e''_{n-1,k}$ using ElGamal decryption function: $r_k = D_k(e''_{n-1,k})$ and publishes r_k.
2. A secret is reconstructed: $r = \prod_{k\in K} r_k^{u_k} \bmod p$.
3. If $r = {E''}_{n-1,0}$, then the DNF is 1. Otherwise, it is 0.

6 Implementation and Efficiency Optimisation

There is an efficiency concern in the operation in Section 5.1, which needs μ inversions, $\sum_{i=1}^{n} M(i)$ multiplications and ν simplified secret sharing operations.

m and t are small integers like 3 or 4. μ is usually not too large in practical applications. The DNF optimisation mechanism guarantees that n is not large if the function is suitable for DNF solution. So the only efficiency concern is about ν, which may be large in some cases. Although each simplified secret sharing operation only cost 1 exponentiation, a large cost is needed when ν is large. An optimised function is proposed as follows to replace the corresponding commitment and sharing functions in Section 5.1 to commit to and share publicly known integers $a_{i,j}$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, M(i)$ in the simplified sharing format when ν is large.

1. A set $A = \bigcup_{i=1,j=1}^{i<n,j<M(i)} \{a_{i,j}\}$ is set up to include all the constant integers involved in all the equations in the DNF in the form (1). Suppose A has a size ν and $A = \{a_1, a_2, \ldots, a_\nu\}$ and the largest integer in A is ρ bits long.
2. For $\tau = 1, 2, \ldots, \rho - 1$, $G_\tau = G_{\tau-1}^2$ are calculated where $G_0 = 1$.
3. For $\iota = 1, 2, \ldots, \nu$, the commitment of a_ι is publicly available as $(\hat{E}_{\iota,0}, \hat{E}_{\iota,1}, \ldots, \hat{E}_{\iota,t-1}) = (\prod_{\tau=0}^{\rho-1} G_\tau^{b_{\iota,\tau+1}} \bmod p,\ 1, 1, \ldots, 1)$ and its share for every A_k is publicly available as $(\hat{s}_{\iota,k}, \hat{r}_{\iota,k}) = (a_\iota, 0)$ where $b_{\iota,\tau}$ is the τ^{th} bit of a_ι.
4. For each $a_{i,j}$, if $a_{i,j} = a_\iota$, then $\hat{E}_{i,j,l} = \hat{E}_{\iota,l}$ for $l = 0, 1, \ldots, t-1$ and $\hat{s}_{i,j,k} = \hat{s}_{\iota,k}$, $\hat{r}_{i,j,k} = \hat{r}_{\iota,k}$ for $k = 1, 2, \ldots, m$.

After this optimisation, cost for committing to and sharing the constant integers in the function is $2(\rho - 1)$ multiplications. Therefore, evaluation of DNFs is efficient. However, until now public verification has not been taken into account. A cautious method to achieve public verifiability called complete public verification procedure is to publicly verify validity of any secret operation. Thus the following proof and verification computations are needed.

1. Each A_k has to publicly prove that his share of $REC(E'_{1,0})$ is encrypted in $c'_{1,k}$ and $e'_{1,k}$ through ZK proof:

$$ZP\ [\ \gamma_{1,k},\ \delta_{1,k},\ s'_{1,k},\ t'_{1,k} \mid c'_{1,k} = (g^{\gamma_{1,k}} \bmod p,\ y_k^{\gamma_{1,k}} g^{s'_{1,k}} \bmod p), \quad (15)$$
$$e'_{1,k} = (g^{\delta_{1,k}} \bmod p,\ y_k^{\delta_{1,k}} h^{r'_{1,k}} \bmod p),\ g^{s'_{1,k}} h^{t'_{1,k}} = \textstyle\prod_{l=0}^{t-1} {E'}_{1,l}^{k^l} \bmod p\ ],$$

   Proof and verification of (15) for $k \in K$ cost $13t$ full length exponentiations, $t(t+5)$ short exponentiations and $t(t+15)$ multiplications.
2. Computation of (12), (13) and (14) must be publicly proved and verified for $i = 2, 3, \ldots, n$ and $k' \in K$ through ZK proof:

$$ZP\ [\ s'_{i,k'},\ t'_{i,k'} \mid {E''}_{i-1,l,k'} = {E''}_{i-2,l}^{s'_{i,k'} u_{k'}} \bmod p \text{ for } l = 0, 1, \ldots, t-1,$$
$${c''}_{i-1,k,k'} = {c''}_{i-2,k}^{s'_{i,k'} u_{k'}} \bmod p \text{ for } k \in K,\ {e''}_{i-1,k,k'} = {e''}_{i-2,k}^{s'_{i,k'} u_{k'}} \bmod p \text{ for } k \in K,$$
$$g^{s'_{i,k'}} h^{t'_{i,k'}} = \textstyle\prod_{l=0}^{t-1} {E'}_{i,l}^{k'^l} \bmod p\ ], \quad (16)$$

   Proof and verification of (12), (13) and (14) for $i = 2, 3, \ldots, n$ and $k' \in K$ cost $(n-1)t(3t+12)$ full length exponentiations, $(n-1)t(t+5)$ short exponentiations and $(n-1)t(4t+16)$ multiplications.

3. Decryption in Step 1 in Section 5.4 must be publicly proved and verified through ZK proof:

$$ZP\ [\ x_k \mid g^{x_k} = y_k,\ a''^{x_k}_{n-1,k} r_k = b''_{n-1,k}\ ] \tag{17}$$

for $k \in K$. (17) can be publicly proved and verified through ZK proof of equality of logarithms [7], which costs $6t$ exponentiations and $3t$ multiplications.
4. As mentioned in Section 4 sharing of the μ secret inputs must be verified, which costs the CPs totally $2m\mu$ exponentiations and $O(m\mu t)$ multiplications. As illustrated in Section 4 if a CP finds an invalid share from an IP, he can publish it and the IP cannot deny it as the communication channel between them is authenticated.

There is a more efficient method to achieve public verifiability. It does not require to publicly verify validity of every secret operation. Instead, it only publicly proves and verifies that $\prod_{i=1}^{n} \sum_{j=1}^{M(i)} (a_{i,j} - m_{i,j}) t_j$ is correctly committed in $E''_{n-1,0}$ and correctly reconstructed. So only the following proof and verification operations are needed.

1. A'_k publicly proves for $i = 2, 3, \ldots, n$ and $k' \in K$

$$ZP\ [\ s'_{i,k'},\ t'_{i,k'} \mid E''_{i-1,0,k'} = E''^{s'_{i,k'} u_{k'}}_{i-2,0} \bmod p,$$
$$g^{s'_{i,k'}} h^{t'_{i,k'}} = \prod_{l=0}^{t-1} E'^{k'^l}_{i,l} \bmod p\ ],$$

verification of which guarantees that $E''_{n-1,0}$ is correctly generated. The proof and verification of (18) are implemented in Figure 1.
2. It is publicly proved and verified that correct shares are used to reconstruct the secret committed in $E''_{n-1,0}$ in Section 5.4. More precisely, after each A_k publishes $s_k = D_k(c''_{n-1,k})$ and $r_k = D_k(e''_{n-1,k})$, it is publicly verified $s_k r_k = \prod_{l=0}^{t-1} E''^{k^l}_{n-1,l} \bmod p$ for $k \in K$.

This efficient public verification procedure (including proof and verification) only costs 6 full length exponentiations, $t^2 + t + 2$ short exponentiations and $t^2 + t + 6$ multiplications, and thus is much more efficient than the complete verification procedure. If it is passed, it is guaranteed that $g^{\prod_{i=1}^{n} \sum_{j=1}^{M(i)} (a_{i,j} - m_{i,j}) t_j}$ is correctly reconstructed to determine the result of the function. If it fails, the complete verification procedure is run and every secret operation is verified until an invalid secret operation is detected. Then the participant responsible for the invalid secret operation is expelled. If a penalty is given to any detected dishonest participant, the participants will usually be honest and in most cases only the efficient verification procedure is needed. Therefore, the DNFs can be efficiently evaluated while public verifiability is achieved.

Theorem 1, Theorem 2 and Theorem 3 illustrate correctness, soundness and privacy of the new scheme respectively. Privacy of the new secure computation protocol with the complete public verification procedure can be proved as well. Due to space limitation, their proof is left to the readers.

1. A'_k publishes $z_{1,i,k'}$ and $z_{2,i,k'}$ where

$$z_{1,i,k'} = ({E''}^{u_{k'}}_{i-2,0})^{v_{1,i,k'}} \bmod p$$
$$z_{2,i,k'} = g^{v_{1,i,k'}} h^{v_{2,i,k'}} \bmod p$$

   and $v_{1,i,k'}$, $v_{2,i,k'}$ are randomly chosen from Z_q.
2. A verifier randomly chooses and publishes a 128 bit integer $u_{i,k'}$.
3. A'_k publishes $w_{1,i,k'}$ and $w_{2,i,k'}$ where

$$w_{1,i,k'} = v_{1,i,k'} - s'_{i,k'} u_{i,k'} \bmod q$$
$$w_{2,i,k'} = v_{2,i,k'} - t'_{i,k'} u_{i,k'} \bmod q$$

Anyone can verify

$$z_{1,i,k'} = ({E''}^{u_{k'}}_{i-2,0})^{w_{1,i,k'}} {E''}^{u_{i,k'}}_{i-1,0,k'} \bmod p$$
$$z_{2,i,k'} = g^{w_{1,i,k'}} h^{w_{2,i,k'}} (\prod_{l=0}^{t-1} {E'}^{k'l}_{i,l})^{u_{i,k'}} \bmod p$$

Fig. 1. ZK proof and verification of (18)

Theorem 1. *The new secure computation protocol is correct.*

Theorem 2. *The new secure computation scheme is sound with the efficient public verification procedure.*

Theorem 3. *The new secure computation protocol is private with the efficient public verification procedure.*

7 A Typical Example and Comparison

Using the DNF generation algorithm in Section 4, the famous millionaire problem (the most popular and typical example in secure computation) is reduced to a simple DNF as follows.

1. m_1 and m_2 are L-bit messages to be compared where $m_1 = (m_{1,1}, m_{1,2}, \ldots, m_{1,L})$, $m_2 = (m_{2,1}, m_{2,2}, \ldots, m_{2,L})$ and $m_{i,j}$ is the j^{th} most important bit of m_i.
2. The DNF to evaluate the function is

$$(m_{1,1} = 1 \wedge m_{2,1} = 0) \vee (m_{1,1} = m_{2,1} \wedge m_{1,2} = 1 \wedge m_{2,2} = 0) \vee$$
$$\ldots \vee (m_{1,1} = m_{2,1} \wedge m_{1,2} = m_{2,2} \wedge \ldots \wedge m_{1,L-1} = m_{2,L-1}$$
$$\wedge m_{1,L} = 1 \wedge m_{2,L} = 0)$$

   which is a simple DNF and can be efficiently evaluated.

In Table 1, the new secure computation scheme is compared with the existing general purpose secure computation schemes. Secure computation techniques only dealing with a special function (like [23, 24, 25, 26]) are not included. As

Table 1. Comparison of properties

Scheme	Sound--ness	Privacy	Flexibility in format	Computation	Communi--cation
[21]	No	Yes	No	$\geq 15KLT$ $= 61440000$	$\geq 37LT + 2T$ $= 148080$
[18]	Yes	Yes	No	$\geq 15KLT$ $= 61440000$	$\geq 37LT + 2T$ $= 148080$
[17]	Yes	Yes	No	average $\geq 4665KL$ $= 477696000$	$\geq 1626L$ $= 162600$
[3]	Yes	Yes	No	average $\geq 4039.5KL$ $= 413644800$	$\geq 1543L$ $= 154300$
[28]	No	Incomplete	No	$> L^4$ $= 100000000$	$\geq 343L^3$ $= 343000000$
New	Yes	Yes	Yes	$1.5K(4L + 40n - 18)+$ $6K'(n-1) + 2L + 44n - 41$ $= 1208663$	$10L + 30n$ -20 $= 1280$

pointed out in Section 1 it is very difficult to precisely estimate the cost of secure computation schemes employing an abstract circuit [8, 9, 10, 11, 17, 18, 21]. They only claimed that a certain evaluation circuit is established to evaluate a function while the concrete algorithm to generate the circuit is not provided and the concrete structure of the circuit is unknown. So there is not an instantiated protocol to be analysed in regard to efficiency in these schemes. Fortunately the concrete cost of [17, 18, 21] can be estimated according to [19], whose result is then used in Table 1. Unfortunately, there is no hint available to the concrete cost of [8, 9, 10, 11], which are thus not included in Table 1. The schemes in [9] and [1] are similar to [17] and [28] respectively, so are not separately listed in Table 1.

For fairness of the comparison, the circumstance of the existing schemes is adopted in the new scheme. For the sake of simplicity and generality, t, the sharing threshold, is set to be 2. As the cost of preliminary operations including set-up of distributed system (distribution of private key[2] or input), input encryption, input validity check and all the public verification operations are not counted in computation efficiency analysis of the existing schemes, their cost is not counted in computation of the new protocols as well. In Table 1, n stands for the number of clauses in the DNF; K is the bit length of a full length integer; K' is the length of challenges in ZK proof primitives and T is the cutting factor in cut-and-choose mechanism. The number of full length multiplications is counted in terms of computation while addition, multiplication of small integers and exponentiations with small base are ignored and exponentiations with full length base are converted into multiplications with a rule: an exponentiation with a x bit exponent is equivalent to $1.5x$ multiplications. In Table 1, transportation of integers with significant length (e.g. 1024 bits long) is counted in regard to

[2] In some secure computation schemes [1, 28], distributed generation of private key is extremely inefficient.

communication. In the example in Table 1, the evaluated function is the millionaire problem and $K = 1024$, $K' = 128$, $L = 100$, $n = 10$ and $T = 40$. It is clearly illustrated in Table 1 that the new secure computation scheme is secure, flexible and much more efficient than the existing secure computation schemes.

8 Conclusion

A new homomorphism based secure multiparty computation scheme with formal security, strong flexibility and high efficiency is proposed. Compared to the other homomorphism based multiparty computation schemes [1, 9, 11, 12, 16, 28], the new scheme is more suitable for functions which can be reduced to polynomial-size DNFs. The new scheme achieves flexibility in input format and supports any input format. Privacy of the new scheme is formally proved in a novel security model, which has independent value. A typical example of evaluating the millionaire problem is given to clearly illustrate advantage of the new scheme in efficiency. The example also demonstrates practicality and applicability of the new scheme.

References

1. Beaver, D.: Minimal-latency secure function evaluation. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 335–350. Springer, Heidelberg (2000)
2. Ben-Or, M., Goldwasser, S., Killian, J., Wigderson, A.: Multi-prover interactive proofs: How to remove intractability assumptions. In: STOC 1988, pp. 113–131
3. Cachin, C., Camenisch, J.: Optimistic fair secure computation (extended abstract). In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 94–112. Springer, Heidelberg (2000)
4. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS 2001, pp. 136–145
5. Canetti, R., Fiege, U., Goldreich, O., Naor, M.: Adaptive secure computation. In: ACM STOC 1996, pp. 143–202
6. Chaum, D., Crepeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC 1988, pp. 11–19
7. Chaum, D., Pedersen, T.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
8. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)
9. Cramer, R., Damgård, I., Nielsen, J.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–299. Springer, Heidelberg (2001)
10. 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)
11. Damgård, I., Nielsen, J.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Heidelberg (2003)

12. Fischlin, M.: A cost-effective pay-per-multiplication comparison method for millionaires. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 457–472. Springer, Heidelberg (2001)
13. Gennaro, R., Rabin, M., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: PODC 1998, pp. 101–111
14. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC 1987, pp. 218–229.
15. Hirt, M., Maurer, U.: Robustness for free in unconditional multi-party computation. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 101–118. Springer, Heidelberg (2001)
16. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure. In: IEEE Symposium on Foundations of Computer Science 2000, pp. 294–304
17. Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Heidelberg (2000)
18. Juels, A., Szydlo, M.: A two-server, sealed-bid auction protocol. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 72–86. Springer, Heidelberg (2003)
19. Kurosawa, K., Ogata, W.: Bit-slice auction circuit. In: Gollmann, D., Karjoth, G., Waidner, M. (eds.) ESORICS 2002. LNCS, vol. 2502, pp. 24–38. Springer, Heidelberg (2002)
20. McKay, C.: Digital Circuit, a Preparation for Microprocessors. Prentice-Hall, Englewood Cliffs (1978)
21. Naor, M., Pinkas, B., Sumner, R.: Privacy perserving auctions and mechanism design. In: ACM Conference on Electronic Commerce 1999, pp. 129–139
22. Pedersen, T.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 129–140. Springer, Heidelberg (1991)
23. Peng, K., Boyd, C., Dawson, E., Lee, B.: An efficient and verifiable solution to the millionaire problem. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 315–330. Springer, Heidelberg (2005)
24. Peng, K., Boyd, C., Dawson, E., Lee, B.: Ciphertext comparison, a new solution to the millionaire problem. In: Qing, S., Mao, W., Lopez, J., Wang, G. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 84–96. Springer, Heidelberg (2005)
25. Peng, K., Boyd, C., Dawson, E., Okamoto, E.: A novel range test. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 247–258. Springer, Heidelberg (2006)
26. Peng, K., Dawson, E.: Range test secure in the active adversary model. In: AISW 2007. ACM International Conference Proceeding Series, vol. 249, pp. 159–162
27. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: ACM STOC 1989, pp. 73–85
28. Sander, T., Young, A., Yung, M.: Non-interactive cryptocomputing for NC^1. In: FOCS 1999, pp. 554–567