J. Software Engineering & Applications, 2010, 3, 709-717
doi:10.4236/jsea.2010.37081 Published Online July 2010 (http://www.SciRP.org/journal/jsea)

Secure Multi-Party Proof and its Applications

Chunming Tang[1,2], Shuhong Gao[3]

[1]School of Mathematics and Information Sciences, Guangzhou University, Guangzhou, China; [2]State Key Laboratory of Information Security, Chinese Academy of Science, Beijing, China; [3]Department of Mathematical Sciences, Clemson University, Clemson, USA. Email: ctang@gzhu.edu.cn

Received April 30th, 2010; revised May 24th, 2010; accepted June 1st, 2010.

ABSTRACT

We define a new type cryptographical model called secure multi-party proof that allows any t players and a verifier to securely compute a function $f(x_1,...,x_t)$: each of the players learns nothing about other players' input and about the value of f, and the verifier obtains the value of f and it's validity but learns nothing about the input of any of the players. It is implemented by a protocol using oblivious transfer and Yao's scrambled circuit. We prove that our protocol is secure if the players and the verifier are semi-honest (i.e. they follow the protocol) and polynomial time bounded. The main applications of our protocol are for electronic voting and electronic bidding.

***Keywords*:** *Multi-Party Proof, Multi-Party Computation, Electronic Voting, Electronic Bidding*

1. Introduction

1.1 Secure Multi-Party Computation and its Disadvantage

In a secure multi-party computation, a set of n parties with private inputs wish to jointly and securely compute a function that depends on the individual inputs of the parties. This computation should be such that each party receives its correct output (correctness), and none of the parties learn anything beyond their prescribed output (privacy). For example, in an election protocol, correctness ensures that no coalition of parties can influence the outcome of the election beyond just voting outcome for their preferred candidate, whereas privacy ensures that no parties learn anything about the individual votes of other parties. Secure multi-party computation can be viewed as the task of carrying out a distributed computation, while protecting honest parties from the malicious manipulation of dishonest (or corrupted) parties.

In all secure multi-party computation, only participants obtain information about the value of the function f computed. In some applications, some party say an arbiter, other than the participants, may want to know the function value and needs to be sure of its validity, yet the arbiter learns nothing about the secret inputs of the participants. Here's a possible scenario. Assume a company needs to appoint a new manager for a department. The administrators of the company hope that the manager is elected by the staff in this department only. The staff could elect a new manager by using an electronic voting protocol from a secure multi-party computation. However, these administrators are usually not in this department, hence they may not be convinced that the election result is valid.

Another main application of secure multi-party computation is the design of electronic bidding protocols. Usually, all participants jointly run a secure multi-party computation protocol and decide the winner; as a result, only all participants know who the winner is. However, the sponsor in any electronic bidding is not a participant; hence the sponsor may not be sure about the winner. Also, in some time the participants may not be allowed to know the winner, as the winner wants to be kept anonymous.

1.2 Our Contributions

We define a new type cryptographical model called secure multi-party proof that allows any t players and a verifier to securely compute a function $f(x_1,...,x_t)$. The model requires the following properties: each of the players learns nothing about other players' input and nor any information about the value of f, and the verifier obtains the value of f and it's validity but learns nothing about the input of any of the players. We implement this model by a protocol using oblivious transfer and

Yao's scrambled circuit. We prove that our protocol is secure if the players and the verifier are semi-honest (*i.e.* they follow the protocol) and polynomial time bounded. Based on our secure multi-party proof, our protocol can be used for electronic voting and electronic bidding.

1.3 Related Work

A great deal of work has been done about secure multi-party computation. Secure computation for two parties was first formulated by Yao [1] in 1982. In [2,3], Lindell and Pinkas gave a complete and explicit proof of Yao's protocol for secure two-party computation. Goldreich, Micali and Wigderson [4] showed how to securely compute any multivariate function (even if malicious adversaries are present); see [5] for complete proof of their results. Ben-Or, Goldwasser and Wigderson [6] (and, independently, Chaum, Crepeau and Damgard [7]) study secure multiparty computation in the secure channels setting. They show that: 1) If the adversary is eavesdropping then there exist $\left(\left\lceil \frac{n}{2} \right\rceil - 1\right)$-secure protocols for computing any function. 2) If the adversary is Byzantine, then any function can be $\left(\left\lceil \frac{n}{3} \right\rceil - 1\right)$-securely computed. Furthermore, they show that these bounds on the number of corruptions are tight. These protocols can be shown secure in the presence of non-adaptive adversaries. Adaptive security (*i.e.*, security in the presence of adaptive adversaries) is provable in certain variants of this setting.

Goldwasser and Levin [8] study the case of Byzantine adversaries where a majority of the parties may be corrupted. Chor and Kushilevitz [9] deal with secure multi-party computation with majority of the parties corrupted in the secure channels setting. Goldreich, Goldwasser and Linial [10] study secure multiparty computation in the presence of insecure channels and computationally unlimited adversaries. Ostrovsky and Yung [11] study secure multiparty computation in the presence of secure channels and mobile adversaries. Micali and Rogaway [12], and also Beaver [13], propose definitions for secure multiparty computation in the secure channels setting in the presence of adaptive adversaries. Other types of secure multi-party computation include adaptively secure multi-party computation [14], almost-everywhere secure computation [15], concurrent secure multi-party computation [16], and fair multi-party computation [17-19] and so on.

1.4 Organization

The paper is organized as follows. We start with some basic definitions and tools in Section 2. Section 3 gives a formal model of secure multi-party proof. Section 4 constructs a secure multi-party proof for any polynomial-time computable function if all participants are semi-honest. Section 5 provides a general method to construct a protocol that can be used for electronic voting and electronic bidding. Finally, Section 6 outlines concluding remarks and future directions.

2. Preliminary and Basic Tools

Let n denote a positive integer. We say that a function $\mu(n)$ is negligible in n (or just negligible) if, for every polynomial $p(n)$, we have $\mu(n) < 1/p(n)$ for all sufficiently large n. Let S be an infinite set and the size of an element $s \in S$ is denoted by $|s|$. Suppose $X = \{X_s\}_{s \in S}$ and $Y = \{Y_s\}_{s \in S}$ are two ensembles of random distributions (or random variables). We say that X and Y are computationally indistinguishable if, for every non-uniform polynomial-time distinguisher D, the absolute difference $|\Pr(D(X_s) = 1) - \Pr(D(Y_s) = 1)|$ is negligible in $|s|$ (for $s \in S$). For a probabilistic machine M, we denote by $M(x)$ the output of M when the input is x. The value $M(x)$ is a probabilistic distribution, as it depends on some random values from an internal random tape used by the machine.

Semi-honest Adversaries vs. Malicious Adversaries. Loosely speaking, the aim of a secure multi-party protocol is to protect honest parties against dishonest behaviors by some other parties. Usually, adversaries are divided into semi-honest and malicious adversaries.

A semi-honest adversary controls one of the parties and follows the protocol specification exactly. However, it may try to learn more information than allowed by the protocol via analyzing the transcript of messages received.

A malicious adversary may arbitrarily deviate from the specified protocol. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, a party may refuse to participate in the protocol or substitute its local input (and use instead a different input). The adversary may also abort the protocol prematurely so that the adversary may obtain its output while the honest party does not.

In this paper, we assume that all parties or adversaries are semi-honest.

Special Symmetric Encryption. In [2], a special symmetric encryption scheme was constructed that has indistinguishable encryption for multiple messages. This means that for any two messages $\overline{x}$ and $\overline{y}$, no polynomial-time adversary can distinguish an encryption of $\overline{x}$ from that of $\overline{y}$.

Definition 1 Let (G, E, D) be a symmetric encryption scheme and let the range of a key k denoted by $R_n(k) = \{E_k(x) : x \in \{0,1\}^n\}$.

1) We say that (G,E,D) has an elusive range if, for every probabilistic polynomial-time machine M and for every polynomial $p(n)$, we have

$$\Pr(M(k) \in R_n(k)) < 1/p(n)$$

for sufficiently large n, where k is a random binary string of length n.

2) We say that (G,E,D) has an efficiently verifiable range if there exists a probabilistic polynomial time machine M such that $M(1^n,k,c)=1$ if and only if $c \in R_n(k)$.

By convention, for every $c \notin R_n(k)$, we let $D_k(c) = \perp$.

In [2], Y. Lindell and B. Pinkas give a simple construction of a special symmetric encryption scheme. Let $F=\{f_k\}$ be a family of pseudorandom functions [11], where $f_k : (0,1)^n \to \{0,1\}^{2n}$ for $k \in \{0,1\}^n$. Then define

$$E_k(x) = (r, f_k(r) \oplus x0^n)$$

where $x \in \{0,1\}^n$, $r \in_R \{0,1\}^n$ and $x0^n$ denotes the concatenation of x and 0^n.

Oblivious Transfer. We will briefly describe the oblivious transfer protocol of [20]. This protocol is secure in the presence of semi-honest adversaries. Our description will be for the case that $x_0, x_1 \in \{0,1\}$; when considering semi-honest adversaries, the general case can be obtained by running the single-bit protocol many times in parallel. It is assumed that there is a family of permutations with trapdoors, so the permutations are one-way functions if the trapdoors are not given. Furthermore, $B(x)$ is a hardcore bit of the one-way functions, so computing $B(x)$ is equivalent to inverting the one-way functions (without using the trapdoors).

Suppose P_1 has two bits $x_0, x_1 \in \{0,1\}$ and P_2 has $\sigma \in \{0,1\}$. The goal is for P_1 to transfer x_σ to P_2 but P_1 does not know which of the two bits was transferred. This is called a 1-out-of-2 oblivious transfer.

Oblivious Transfer Protocol

1) P_1 randomly chooses a permutation-trapdoor pair (E,t) from a family of trapdoor permutations. P_1 sends E (but not the trapdoor t) to P_2.

2) P_2 chooses a random v_σ in the domain of E and computes $\omega_\sigma = f(v_\sigma)$. In addition, P_2 chooses a random $\omega_{1-\sigma}$ in the range of E. P_2 sends (ω_0, ω_1) to P_1.

3) P_1 uses the trapdoor t and computes $v_0 = E^{-1}(\omega_0)$ and $v_1 = E^{-1}(\omega_1)$. Then computes $b_0 = B(v_0) \oplus x_0$ and $b_1 = B(v_1) \oplus x_1$, where B is a hard-core bit of E. Finally, P_1 sends (b_0, b_1) to P_2.

4) P_2 computes $x_\sigma = B(v_\sigma) \oplus b_\sigma$, which is the bit transferred to P_2 by P_1.

It was proven in [21] that the above protocol is secure if both of P_1 and P_2 are semi-honest.

3. Definition of Secure Multi-Party Proof

Suppose there are t players $P_1, P_2, \cdots, P_t$ with secret inputs $x_1, x_2, \cdots, x_t$, respectively, and a verifier V. We assume that all the players and the verifier are computationally bounded. In addition, assume that $f(x_1, x_2, \cdots, x_t)$ is a polynomial-time computable function, so it has a polynomial size circuit.

Definition 2. A multi-party Proof consists of two sub-protocols:

Computation sub-protocol: It is an ordinary multi-party computation among the players $P_1, P_2, \cdots, P_t$. The goal is to compute a value for each player, say m_i for P_i for $1 \le i \le t$. Each value m_i depends on x_i and a random binary string r_i, both of them are kept secret by player P_i for $1 \le i \le t$. Each player does not gain any information about $f(x_1, x_2, \cdots, x_t)$ and any information on other players' inputs. Then each player P_i sends secretly m_i to V, $1 \le i \le t$.

Proof sub-protocol: In this sub-protocol, the verifier V computes the value $f(x_1, x_2, \cdots, x_t)$ from $m_1, m_2, \cdots, m_t$ and verifies its validity.

When defining security of multi-party proof, we have to consider the security of each of the two sub-protocols. The computation sub-protocol is an ordinary multi-party computation which allows a set of mutually distrusting parties to compute a function in a distributed way while guaranteeing (to the extent possible) the privacy of their local inputs and the correctness of the outputs. To be more exact, security is typically formulated by comparing a real execution of the protocol to an ideal execution where the parties just send their inputs to a trusted party and receive back their outputs. A real protocol is said to be secure if an adversary can do no more harm in a real execution than in an ideal execution (which is secure by definition). The main security properties that have been identified, and are implied by this formulation, are privacy (parties learn nothing more than their own output) and correctness (the outputs are correctly computed).

In the proof sub-protocol, the verifier V will learn nothing beyond the value of $f(x_1, x_2, \cdots x_t)$ and its va-

lidity, that is, V only obtains $f(x_1, x_2, \cdots x_t)$ and its validity. In another word, V does not gain any information about $x_1, x_2, \cdots, x_t$ yet is convinced that the values of $x_1, x_2, \cdots, x_t$ used in computing $f(x_1, x_2, \cdots, x_t)$ are the same as claimed, and furthermore, the verifier obtains the correct function value $x_1, x_2, \cdots, x_t$. This is defined more formally as follows.

Definition 3 (Security of Multi-Party Proof)

A multi-party proof is secure if it satisfies the following properties:

1) **Correctness**: Suppose each player P_1 (for $i = 1, 2, \cdots, t$) chooses random binary string r_i of length n. Let y_0 be the final value computed by V from $m_1, m_2, \cdots, m_t$. Then y_0 is equal to $f(x_1, x_2, \cdots, x_t)$ with high probability, that is, the probability $\Pr(y_0 \neq f(x_1, x_2, \cdots, x_t))$ is negligible as a function of n.

2) **Privacy**: For each player $P_i, 1 \le i \le t$, let M_i be all the message that P_i obtains from all other players in the computation sub-protocol. The protocol is said to have privacy for all the players if, for each $1 \le i \le t$, there exists a bounded probabilistic polynomial time simulator S_i such that M_i is indistinguishable from the output $S_i(1^n)$ of the simulator S_i.

For the verifier V, let $M_v = (m_1, m_2, \cdots, m_t)$ denote all the messages received from the players. We say that the protocol has privacy for V if there exists a bounded probabilistic polynomial time simulator S_v so that M_v is indistinguishable with the output $S_v(1^n)$ of the simulator S_v.

Also, none of the players learn anything about the function value computed.

3) Validity: The validity includes the two following properties:

a) V can verify with high probability that the value m_i is correctly computed from the claimed value of x_i, $1 \le i \le t$, all of which are kept secret from V.

b) V can verify the correctness of y_0. Assume the function f is given as a boolean circuit C. Then V can verify every gate's computation from the input wires to the output wires.

4. Construction of Secure Multi-Party Proof

In this section, we will construct a secure multi-party proof for any polynomial-time computable function $f(x_1, x_2, \cdots x_t)$ if all players $P_1, \cdots, P_n$ are semi-honest.

4.1 Two-Party Computation Secure against Semi-Honest Adversaries

We firstly describe the construction of secure two-party computation (for semi-honest adversaries) due to Yao [1]. We follow the description by [2,3] where it is proven to be secure against semi-honest adversaries.

Let C be a Boolean circuit that receives two inputs $x, y \in \{0,1\}^n$ and outputs $C(x, y) \in \{0,1\}^n$ (for simplicity, we assume that the input length, output length and the security parameter are all n). We also assume that C has the property that every gate with a circuit-output wire has no other outgoing wires. We begin by describing the construction of a single garbled gate g in C. The circuit C is Boolean, and therefore any gate is represented by a function $g: \{0,1\} \times \{0,1\} \to \{0,1\}$. Let the two input wires to g be labeled ω_1 and ω_2. g may have several outgoing wires, but all of them are labeled by the same symbol ω_3. Furthermore, let $k_1^0, k_1^1, k_2^0, k_2^1, k_3^0, k_3^1$ be six random keys obtained by independently invoking the keygeneration algorithm $G(1^n)$; for simplicity, assume that these keys are also of length n. Intuitively, we wish to be able to compute $k_3^{g(\alpha,\beta)}$ from k_1^α and k_2^β, without revealing any of the other three values, $k_3^{g(1-\alpha,\beta)}$, $k_3^{g(\alpha,1-\beta)}$, and $k_3^{g(1-\alpha,1-\beta)}$. The garbled gate g is defined by the following four values

$$c_{0,0} = E_{k_1^0}\left(E_{k_2^0}\left(g_3^{g(0,0)}\right)\right)$$

$$c_{0,1} = E_{k_1^0}\left(E_{k_2^1}\left(g_3^{g(0,1)}\right)\right)$$

$$c_{1,0} = E_{k_1^1}\left(E_{k_2^0}\left(g_3^{g(1,0)}\right)\right)$$

$$c_{1,1} = E_{k_1^1}\left(E_{k_2^1}\left(g_3^{g(1,1)}\right)\right)$$

where E is from a private key encryption scheme (G, E, D) that has indistinguishable encryptions for multiple messages, and has an elusive efficiently verifiable range [2,3]. The actual gate is defined by a random permutation of the above values, denoted as c_0, c_1, c_2, c_3; from here on we call them the garbled of gate g. Notice that given k_1^α and k_2^β, and these values c_0, c_1, c_2, c_3, it is possible to compute the output of the gate $k_3^{g(\alpha,\beta)}$ as follows. For every i, computes $D_{k_2^\beta}\left(D_{k_1^\alpha}(c_i)\right)$. If there are more than one decryption then returns a non-$\perp$ value, then output abort. Otherwise, define k_3^γ to be the

only non-$\perp$ value that is obtained. (Notice that if only a single non-$\perp$ value is obtained, then this will be $k_3^{g(\alpha,\beta)}$ because it is encrypted under the given keys k_1^{α} and k_2^{β}. Later we will show that except with negligible probability, only one non-$\perp$ value is indeed obtained.)

We are now ready to show how to construct the entire garbled circuit. Let m be the number of wires in the circuit C, and let $\omega_1,\cdots,\omega_m$ be labels of these wires. These labels are all chosen uniquely with the following exception: if ω_i and ω_j are both output wires from the same gate g, then $\omega_i=\omega_j$ (this occurs if the fan-out of g is greater than one). Likewise, if an input bit enters more than one gate, then all circuit-input wires associated with this bit will have the same label. Next, for every label ω_i, choose two independent keys $k_i^0,k_i^1\leftarrow G(1^n)$; we stress that all of these keys are chosen independently of the others. Now, given these keys, the four garbled values of each gate are computed as described above and the results are permuted randomly. Finally, the output or decryption tables of the garbled circuit are computed. These tables simply consist of the values $(0,k_i^0)$ and $(1,k_i^1)$ where ω_i is a circuit-output wire. (Alternatively, output gates can just compute 0 or 1 directly. That is, in an output gate, one can define $c_{\alpha,\beta}=E_{k_1^{\alpha}}\left(E_{k_2^{\beta}}\left(g(\alpha,\beta)\right)\right)$ for every $\alpha,\beta\in\{0,1\}$).

The entire garbled circuit of C, denoted $G(C)$, consists of the garbled table for each gate and the output tables. We note that the structure of C is given, and the garbled version of C is simply defined by specifying the output tables and the garbled table that belongs to each gate. This completes the description of the garbled circuit.

Let $x=x_1,\cdots,x_n$ and $y=y_1,\cdots,y_n$ be two n-bit inputs for C. Furthermore, let $\omega_1,\cdots,\omega_n$ be the input labels corresponding to x, and let $\omega_{n+1},\cdots,\omega_{2n}$ be the input labels corresponding to y. It is shown in [2] that given the garbled circuit $G(C)$ and the strings $k_1^{x_1},\cdots,k_n^{x_n},k_{n+1}^{y_1},\cdots,k_{2n}^{y_n}$, it is possible to compute $C(x,y)$, except with negligible probability. The complete protocol is as followed.

Protocol 1 (Yao's two-party protocol):

1) **Inputs**: P_1 has $x\in\{0,1\}^n$ and P_2 has $y\in\{0,1\}^n$.

2) **Auxiliary input**: A boolean circuit C such that for every $x,y\in\{0,1\}^n$ it holds that $C(x,y)=f(x,y)$, where $f:\{0,1\}^n\times\{0,1\}^n\rightarrow\{0,1\}^n$. We require that C is such that every gate with a circuit-output wire has no other outgoing wires.

3) The protocol

a) P_1 constructs the garbled circuit $G(C)$ as described in above, and sends it to P_2.

b) Let $\omega_1,\cdots,\omega_n$ be the circuit-input wires corresponding to x, and let $\omega_{n+1},\cdots,\omega_{2n}$ be the circuit-input wires corresponding to y. Then,

i. P_1 sends P_2 the strings $k_1^{x_1},\cdots,k_n^{x_n}$.

ii. For every i,P_i and P_2 execute a 1-out-of-2 oblivious transfer protocol in which P_1 input equals (k_{n+i}^0,k_{n+i}^1) and P_2's input equals y_i.

The above oblivious transfers can all be run in parallel.

c) Following the above, P_2 has obtained the garbled circuit and $2n$ keys corresponding to the $2n$ input wires to C. Party P_2 then computes using the garbled circuit, as described above, obtaining $f(x)$. P_2 then sends the value $f(x)$ to P_1, and they both output this value.

Assume that the oblivious transfer protocol is secure in the presence of static semi-honest adversaries, and that the encryption scheme has indistinguishable encryptions for multiple messages, and has an elusive and efficiently verifiable range. Then it is proved in [2] that Protocol 1 securely computes f in the presence of static semi-honest adversaries.

4.2 Secure Multi-Party Proof against Semi-Honest Adversaries

In this section, we will construct multi-party proof secure against semi-honest adversaries for any polynomial time computable function. We firstly construct a secure multi-party proof between two parties, then generalize it to a secure multi-party proof for more than two participants.

4.2.1 Secure Two-Party Proof against Semi-Honest Adversaries

Assume $f(x)$ is a polynomial computable function, so there exists a polynomial size boolean circuit C such that for every $x,y\in\{0,1\}^n$ it holds that $C(x,y)=f(x,y)$. Both $f(x,y)$ and C are public. We claim that the following protocol is a secure two-party proof for f.

Protocol 2:

1) **Input**: P_1 has $x\in\{0,1\}^n$ and P_2 has $y\in\{0,1\}^n$.

2) **Set-up**:

a) Let $\omega_1,\cdots,\omega_n$ be the circuit-input wires corresponding to x, and let $\omega_{n+1},\cdots,\omega_{2n}$ be the circuit-input wires corresponding to y. Let s be the number of gates in C excluding its circuit-input gates and circuit-output gates

b) By running key generating algorithm G for $4n+2s$ times, P_1 generates $4n+2s$ independent strings

$$k_1^0,k_1^1,\cdots,k_{2n}^0,k_{2n}^1,k_{2n+1}^0,k_{2n+1}^1,\cdots,k_{2n+s}^0,k_{2n+s}^1,$$

then constructs a garbled computation table for every gate by using the special symmetric encryption described in Section 2, and obtains a garbled circuit $G(C)$ which consists of the garbled table for each gate and the output tables. Finally, P_1 publicizes $G(C)$, and keeps k_1^0, $k_1^1,\cdots,k_{2n+s}^0,k_{2n+s}^1$ secret.

3) Computation Sub-protocol:

a) For every $1\le i\le n$, P_1 and P_2 execute a 1-out-of-2 oblivious transfer protocol in which P_1's input equals $\left(k_{n+i}^0,k_{n+i}^1\right)$ and P_2's input equals y_i.

b) The above oblivious transfers can be run by using the oblivious transfer protocol in Section 2 and can be done in parallel.

c) P_1 sends V the strings $k_1^{x_1},\cdots,k_n^{x_n}$.

d) P_2 sends V the strings $k_{n+1}^{y_1},\cdots,k_{2n}^{y_n}$.

4) Proof Sub-protocol:

V has obtained the garbled circuit $G(C)$ and $2n$ keys corresponding to the $2n$ input wires to C. V then computes and obtains $f(x,y)$ via $G(C)$.

Proof: **Correctness**. It is correct by the design of the garbled circuit.

Privacy. Assume that P_1 constructs the garbled circuit $G(C)$ for boolean circuit C. According to the oblivious transfer protocol in Section 2, all messages P_1 obtains are $\left((\omega_{11},\omega_{12}),(\omega_{21},\omega_{22}),\cdots,(\omega_{n1},\omega_{n2})\right)$ from n 1-out-of-2 oblivious transfer protocols between P_1 and P_2. In the transfer protocol, every ω_{ij} is chosen at random. The simulator M for P_1 is just a pseudorandom generator, see for example [12,16]. Let $M\left(1^n\right)$ denotes the output of M which has length $2nk\left(k=\left|\omega_{ij}\right|\right)$. Then $((\omega_{11},\omega_{12}),(\omega_{21},\omega_{22}),\cdots,(\omega_{n1},\omega_{n2}))$ and $M\left(1^n\right)$ is computational indistinguishable. That is, privacy for P_1 is satisfied.

Now we consider privacy for P_2. All messages P_2 obtains are $((E_1,b_{11},b_{12}),(E_2,b_{21},b_{22}),\cdots,(E_n,b_{n1},b_{n2}))$, where E_i is a permutation-trapdoor and pair (b_{i1},b_{i2}) is random for $i=1,2,\cdots,n$. Hence, privacy for P_2 is satisfied by using a pseudorandom generator as a simulator.

The privacy for both P_1 and P_2 implies that P_1 and P_2 do not obtain any information on the other player's input.

Next we consider privacy for V in computation sub-protocol. All messages V obtains are garbled circuit $G(C),k_1^{x_1},k_2^{x_2},\cdots,k_n^{x_n}$ from P_1, and $k_{n+1}^{y_1},k_{n+2}^{y_2},\cdots,k_{2n}^{y_n}$ from P_2. The symmetric encryption algorithm used in $G(C)$ is the special symmetric encryption scheme in Section 2. Because the output of the symmetric encryption algorithm is random and both $k_1^{x_1},k_2^{x_2},\cdots,k_n^{x_n}$ and $k_{n+1}^{y_1},k_{n+2}^{y_2},\cdots,k_{2n}^{y_n}$ are random, privacy for V is satisfied by using a pseudorandom generator as a simulator.

Finally, each player learns nothing about the function value computed by V, since the player know nothing about what other players send to the verifier.

Validity. Because all participants are semi-honest, the validity of all the value m_i computed by P_i from x_i holds automatically. To see the correctness of the value $f(x,y)$, note that V obtains $G(C)$ and $k_1^{x_1},k_2^{x_2},\cdots,k_n^{x_n}$ from P_1, and $k_{n+1}^{y_1},k_{n+2}^{y_2},\cdots,k_{2n}^{y_n}$ from P_2. Hence, according to properties of special symmetric encryption, he can compute $f(x,y)$ and verifies its validity by verifying every gate's computation from the circuit-input wires to circuit-output wires.

4.2.2 Secure Multi-Party Proof against Semi-Honest Adversaries

In this section, we will generalize two-party proof to multi-party proof. Assume $f(x_1,\cdots,x_t)$ is a polynomial time computable function so there exists a boolean circuit C of polynomial size such that, for every $x_1,\cdots,x_t\in\{0,1\}^n$, it holds that $C(x_1,\cdots,x_t)=f(x_1,\cdots,x_t)$. Both $f(x_1,\cdots,x_t)$ and C are made public.

Protocol 3:

1) **Inputs**: P_i has $x_i\in\{0,1\}^n$ for $i=1,2,\cdots,t$.

2) **Set-up**:

a) Let $\omega_{i1},\omega_{i2},\cdots,\omega_{in}$ be the circuit-input wires corresponding to x_i for $i=1,2,\cdots,t$. Let s denote the

number of gates in the circuit C excluding its circuit-input gates and circuit-output gates.

b) By running key generating algorithm G for $2tn+2s$ times, P_1 generates $2tn+2s$ independent strings $k_1^0, k_1^1, \cdots, k_{tn}^0, k_{tn}^1, \cdots, k_{tn+s}^0, k_{tn+s}^1$ constructs a garbled computation table for every gate by using special symmetric encryption in Section 2, and obtains a garbled circuit $G(C)$ which consists of the garbled table for each gate and the output tables. Finally, P_1 publicizes $G(C)$, and keeps $k_1^0, k_1^1, \cdots, k_{tn}^0, k_{tn}^1, \cdots, k_{tn+s}^0, k_{tn+s}^1$ secret.

3) **Computation Sub-protocol:**

a) For every $i(=2,\cdots,t)$, P_1 and P_i execute a 1-out-of-2 oblivious transfer protocol in which P_1's input equals $\left(k_{ij}^0, k_{ij}^1\right)$ and P_i's input equals x_{ij}, for $j = 1,2,\cdots,n$.

b) The above oblivious transfers can be run by using the oblivious transfer protocol in Section 2 and can be done in parallel.

c) P_i sends V the strings $k_{i1}^{x_{i1}}, \cdots, k_{in}^{x_{in}}$ for $i = 1,2,\cdots,t$.

4) **Proof Sub-protocol**:

V has obtained the garbled circuit $G(C)$ and tn keys corresponding to the tn input wires to C. V computes and obtains $f(x_1 \cdots, x_t)$ via $G(C)$.

Proof: **Correctness**. It follows from the design of the protocol.

Privacy. Assume that P_1 constructs the garbled circuit $G(C)$ for boolean circuit C. Then, according to the oblivious transfer protocol in Section 2, all the messages P_1 sees are $(\left(\omega_{11}^i, \omega_{12}^i\right), \left(\omega_{21}^i, \omega_{22}^i\right), \left(\omega_{n1}^i, \omega_{n2}^i\right))$ from n 1-out-of-2 oblivious transfer protocols between P_1 and $P_i(i=2,\cdots,t)$. By the design of the oblivious transfer protocol, every $\omega_{jk}^i (i=2,\cdots,t, j=1,2,\cdots,n)$ is random. The simulator M for P_1 is just a pseudorandom generator, say the one constructed in [12,16]. Let $M(1^n)$ denote the output of M, whose output length is $2(t-1)nk\left(k=\left|\omega_{ij}^i\right|\right)$.

Then $\left\{\left(\left(\omega_{11}^2, \omega_{12}^2\right), \left(\omega_{21}^2, \omega_{22}^2\right), \cdots, \left(\omega_{n1}^2, \omega_{n2}^2\right)\right), \cdots, \left(\left(\omega_{11}^t, \omega_{12}^t\right), \left(\omega_{21}^t, \omega_{22}^t\right), \cdots, \left(\omega_{n1}^t, \omega_{n2}^t\right)\right)\right\}$ and $M(1^n)$ is computational indistinguishable. That is, privacy for P_1 is satisfied.

We consider privacy for every $P_i(i=2,\cdots,t)$. All messages P_i obtains are $\left(\left(E_1^i, b_{11}^i, b_{12}^i\right), \left(E_2^i, b_{21}^i, b_{22}^i\right), \cdots, \left(E_n^i, b_{n1}^i, b_{n2}^i\right)\right)$, where each E_j^i is a permutation-trapdoor and the pair $\left(b_{j1}^i, b_{j2}^i\right)$ is random for $j=1,2,\cdots,n$. Hence, privacy for P_i is satisfied by using a pseudorandom generator as a simulator.

The privacy for all $P_1, P_2, \cdots, P_t$ implies that every P_i obtains no information on other players input during the computation sub-protocol.

Now we consider privacy for V in computation sub-protocol. All messages V obtains are garbled circuit $G(C)$ and $k_{11}^{x_{11}}, k_{12}^{x_{12}}, \cdots, k_{1n}^{x_{1n}}$ from P_1, and $k_{i1}^{x_{i1}}, k_{i2}^{x_{i2}}, \cdots, k_{in}^{x_{in}}$ from P_i, $i=2,\cdots,t$. The symmetric encryption algorithm used in $G(C)$ is the special symmetric encryption scheme in Section 2. Because output of the symmetric encryption algorithm is random and every $k_{ij}^{x_{ij}} (i=1,2,\cdots,t, j=1,2,\cdots,n)$ is random, privacy for V is satisfied by using a pseudorandom generator as a simulator.

Finally, each player learns nothing about the function value computed by V, since the player know nothing about what other players send to the verifier.

Validity. Because all participants are semi-honest, part (a) of the validity holds automatically. For part (b) of the validity, V obtains $G(C)$ and $k_{11}^{x_{11}}, k_{12}^{x_{12}}, \cdots, k_{1n}^{x_{1n}}$ from P_1, and gains $k_{i1}^{x_{i1}}, k_{i2}^{x_{i2}}, \cdots, k_{in}^{x_{in}}$ from P_i, $i=2,\cdots,t$. Hence, according to the properties of the special symmetric encryption, V can compute $f(x_1, x_2, \cdots, x_t)$ and verifies its validity by verifying every gate's computation from the circuit-input wires to circuit-output wires.

5. Electronic Protocols Based on Secure Multi-Party Proof

Protocol 3 can be used in many applications where the verifier has no influence on the functions to be computed, yet he/she wants to learn the function values, however, Protocols 1 and 2 cannot be used because there are only two participants. The arbitrator is assured of the correctness and validity of the function value computed. Two important applications we have in mind are electronic voting and electronic bidding.

Assume $f(x_1, \cdots, x_t)$ is a polynomial time computable function used in an application.

For electronic voting, the function is a sum: $f(x_1, x_2, \cdots, x_t) = x_1 + x_2 + \cdots + x_t$ the total count for votes of a candidate. For electronic bidding, we may

have $f(x_1, x_2, \cdots, x_t) = \max\{x_1, x_2, \cdots, x_t\}$ where x_i is the bid from P_i, $1 \le i \le t$. For these functions there is polynomial sized circuit C for computing them. Both the function $f(x_1, \cdots, x_t)$ and its circuit C are public information. Then our Protocol 3 above can be applied to both of these cases, and it is secure if the participants and the verifier are semi-honest.

6. Conclusions

We defined a new type cryptographical model called secure multi-party proof that allows any t players and a verifier to securely compute a function with t variables. We presented a protocol that is secure when all the participants and the verifier are semi-honest. Our protocol can be used for electronic voting and electronic bidding.

The main difference between multi-party computation and multi-party proof is that, in the former case only participants know the output or partial output, while in the later case only a designated verifier learns the final output. The latter is more practicable in some important situations, for example, in an electronic bidding, where an arbiter is only a verifier but not a participant.

It should be noted, however, that our protocol is not secure against malicious adversaries.

Based on non-interactive zero-knowledge proof, it is possible to construct secure multi-party proof against malicious adversaries. However, these protocols are inefficient because of complexity of zero-knowledge proof. In future work, we intend to construct secure multi-party proof for any polynomial-time computable function $f(x_1, x_2, \cdots, x_t)$ from tools other than zero-knowledge proof.

7. Acknowledgements

This work was supported by Foundation of National Natural Science (China) (10871222) and Opening Foundation of Key Lab of Cryptological Technology and Information Security Ministry of Education in Shandong University.

REFERENCES

[1] A. Yao, "Protocols for Secure Computation," *Proceedings of the* 23*rd Annual Symposium on Foundations of Computer Science*, Chicago, November 1982, pp. 160-164.

[2] Y. Lindell and B. Pinkas, "A Proof of Yao's Protocol for Secure Two-Party Computation," *Journal of Cryptology*, Vol. 22, No. 2, April 2009, pp. 161-188.

[3] Y. Lindell and B. Pinkas, "An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries," *Advances in Cryptology-Eurocrypt* 2007, LNCS 4515, Barcelona, May 2007, pp. 52-78.

[4] O. Goldreich, S. Micali and A. Wigderson, "How to Play Any Mental Game—A Completeness Theorem for Protocols with Honest Majority," *Proceedings of the* 19*th Annual ACM Symposium on Theory of Computing*, New York, 1987, pp. 218-229.

[5] O. Goldreich, "Foundations of Cryptography: Volumne 2—Basic Applications," Cambridge University Press, 2004.

[6] M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," *Proceedings of the* 20*th Annual ACM Symposium on Theory of Computing*, Chicago, 1988, pp. 1-10.

[7] D. Chaum, C. Crepeau and I. Damgard, "Multiparty Unconditionally Secure Protocols," *Proceedings of the* 20*th Annual ACM Symposium on Theory of Computing*, Chicago, 1988, pp. 11-19.

[8] S. Goldwasser and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority," *Advances of Cryptology-Crypto*'90, LNCS 537, Santa Barbara, California, August 1990, pp. 77-93.

[9] B. Chor and E. Kushilevitz, "A Zero-One Law for Boolean Privacy," *SIAM Journal on Discrete Mathematics*, Vol. 4, No. 1, February 1991, pp. 36-47.

[10] O. Goldreich, S. Goldwasser and N. Linial, "Fault-Tolerant Computation in the Full Information Model," *SIAM Journal on Computing*, Vol. 27, No. 2, 1991, pp. 506-544.

[11] R. Ostrovsky and M. Yung, "How to Withstand Mobile Virus Attacks," *Proceedings of the* 10*th Annual ACM Symposium on Principles of Distributed Computing*, Montreal, August 1991, pp. 51-59.

[12] S. Micali and P. Rogaway, "Secure Computation," *Advances of Cryptology-Crypto*'91, LNCS 576, Santa Barbara, California, August 1991, pp. 392-404.

[13] D. Beaver, "Foundations of Secure Interactive Computing," *Advances of Cryptology-Crypto*'91, LNCS 576, Santa Barbara, California, August 1991, pp. 377-391.

[14] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Multi-Party Computation," *Proceedings of the* 28*th Annual ACM Symposium on Theory of Computing*, Philadelphia, 1996, pp. 639-648.

[15] J. A. Garay and R. Ostrovsky, "Almost-Everywhere Secure Computation," *Advances of Cryptology-Eurocrypt* 2008, LNCS 4965, Istanbul, April 2008, pp. 307-323.

[16] R. Pass, "Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority," *Proceedings of the* 36*th Annual ACM Symposium on Theory of Computing*, Chicago, 2004, pp. 232-241.

[17] C. Cachin and J. Camenisch, "Optimistic Fair Secure Computation," *Advances of Cryptology-Crypto*'00, LNCS 1880, Santa Barbara, California, August 2000, pp. 93-111.

[18] S. D. Gordon, C. Hazay, J. Katz and Y. Lindell, "Complete Fairness in Secure Two-Party Computation," *Proceedings of the* 40*th Annual ACM Symposium on Theory*

of Computing, Victoria, 2008, pp. 413-422.

[19] B. Pinkas, "Fair Secure Two-Party Computation," *Advance in Cryptology-Eurocrypt* 2003, LNCS 2656, Warsaw, May 2003, pp. 87-106.

[20] S. Even, O. Goldreich and A. Lempel, "A Randomized Protocol for Signing Contracts," *Communications of the ACM*, Vol. 28, No. 6, 1985, pp. 637-647.

[21] O. Goldreich, S. Goldwasser and S. Micali, "How to Construct Random Functions," *Journal of the ACM*, Vol. 33, No. 4, 1986, pp. 792-807.

[22] O. Goldreich, H. Krawcyzk and M. Luby, "On the Existence of Pseudorandom Generators," *SIAM Journal on Computing*, Vol. 22, No. 6, 1993, pp. 1163-1175.

[23] J. Hastad, R. Impagliazzo, L. A. Levin and M. Luby, "Construction of a Psedorandom Generator from Any One-Way Function," *SIAM Journal on Computing*, Vol. 28, No. 4, 1999, pp. 1364-1396.

