

Transforming semi-honest protocols to ensure accountability

Wei Jiang ^{a,*}, Chris Clifton ^a, Murat Kantarcioglu ^b

^a Department of Computer Science, Purdue University, West Lafayette, IN 47907, United States

^b Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083, United States

Available online 26 July 2007

Abstract

The secure multi-party computation (SMC) model provides means for balancing the use and confidentiality of distributed data. This is especially important in the field of privacy-preserving data mining (PPDM). Increasing security concerns have led to a surge in work on practical secure multi-party computation protocols. However, most are only proven secure under the semi-honest model, and security under this adversary model is insufficient for many PPDM applications. SMC protocols under the malicious adversary model generally have impractically high complexities for PPDM. We propose an accountable computing (AC) framework that enables liability for privacy compromise to be assigned to the responsible party without the complexity and cost of an SMC-protocol under the malicious model. We show how to transform a circuit-based semi-honest two-party protocol into a protocol satisfying the AC-framework. The transformations are simple and efficient. At the same time, the verification phase of the transformed protocol is capable of detecting any malicious behaviors that can be prevented under the malicious model.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Secure multiparty computation; Privacy-preserving

1. Introduction

Privacy and data utility are often perceived to be at odds. An omniscient data source would have many benefits, particularly in support of data mining. On the other hand, an omniscient data source eases misuse, such as the growing problem of identity theft. To prevent misuse of data, there has been a recent surge in laws mandating protection of confidential data, such as the European Community privacy standards [1], US healthcare laws [2], and California SB1386. However, this protection comes with a real cost through both added security expenditure and penalties and costs associated with disclosure. For example, CardSystems was terminated by Visa and American Express after having credit card information stolen [3]. ChoicePoint stock lost 20% of its value in the month following their disclosure of information theft. Such public relations costs can be enormous and could potentially kill a company. From lessons learned in practice, what we need is the ability to compute the desired “beneficial outcome” of sharing data for mining without having to actually

* Corresponding author.

E-mail addresses: wjiang@cs.purdue.edu (W. Jiang), clifton@cs.purdue.edu (C. Clifton), muratk@utdallas.edu (M. Kantarcioglu).

share or disclose data. We can maintain the security provided by separation of control while still obtaining the benefits of a global data source.

Secure multi-party computation (SMC) [4–6] has recently emerged as an answer to this problem. Informally, if a protocol meets the SMC definitions, the participating parties learn only the final result and whatever can be inferred from the final result and their own inputs. A simple example is Yao's millionaire problem [5]: two millionaires want to learn who is richer without disclosing their actual wealth to each other. Recognizing this, the research community has developed many SMC protocols, for applications as diverse as forecasting [7], data analysis [8] and auctions [9].¹ With such a protocol, liability for disclosure of private information falls squarely on the original custodian of that information, as the data is not disclosed during the protocol and thus could not have been disclosed by other parties.

Formal definitions of SMC exist for two adversary models: semi-honest and malicious. In the semi-honest model, it is assumed that each party follows the protocol. However, after the protocol is complete, the adversary may attempt to compute additional information from the messages received during execution. In the malicious model, a party can diverge arbitrarily from normal execution of the protocol. It has been proven that for any polynomial-time algorithm, there exists a polynomial-time secure protocol that achieves the same functionality under either the semi-honest or the malicious model [4]. Nevertheless, most practical algorithms developed have only been proven secure under the semi-honest model. While not a proof, this certainly gives evidence that achieving security against a malicious adversary adds significant complexity and expense.

An SMC-protocol secure under the semi-honest model (or an SSMC-protocol) rarely provides sufficient security for practical applications. A dishonest party could learn private information by not following the protocol, then disclose that information, with blame falling on the innocent original data custodian. (Alternatively, the original data custodian could disclose the private data, then *claim* the other party was dishonest, learned and disclosed the data, and should share liability.) For example, two competing transportation companies want to mine useful patterns among their customers to decide if they can collaborate. Assume there exists an SSMC-protocol that searches for possible overlapping patterns. It is difficult to convince the companies of the need for the protocol if they trust each other; without trust (to follow the protocol correctly) a semi-honest protocol provides no guarantees. However, if cheating can be prevented or caught, contractual penalties can be used to overcome trust issues and enable collaboration. An SMC-protocol secure under the malicious model (or an MSMC-protocol) generally provides such a guarantee, but the complexity of an MSMC-protocol commonly prevents it from being adopted in practice.

Fortunately, our proposed AC-framework can be utilized to design more practical and efficient protocols. The idea behind the AC-framework is that a party who correctly followed the protocol can be proven to have done so and consequently prove that it did not know (and thus could not have disclosed) private data. This provides substantial practical utilities over a semi-honest protocol. In addition, although a malicious adversary participating in an AC-protocol may learn things that they should not know and damage the result, such a behavior could be detected under the AC-framework. Furthermore, since the AC-framework does not need to prevent disclosure to a malicious adversary, protocols can be less complex. In particular, much of the cost can be pushed to a verification phase which needs only be run to expose the culprit when disclosure is detected or auditing is performed to verify honest behaviors among collaborating parties. This enables protocols that approach the efficiency of semi-honest protocols and leads to many practical applications for which the semi-honest protocols are insufficient.

The goal of this paper is to show that without sacrificing its utility and efficiency, functionality computable under a two-party SSMC-protocol can be computed under the AC-framework. Although an MSMC-protocol directly prevents malicious behaviors, the verification phase of the AC-transformed protocol is at least able to detect any malicious behaviors that can be prevented under the malicious adversary model. The paper is organized as follows: Section 2 presents current state of the art from the literature of SMC. Section 3 introduces a simplified version of the AC-framework. Section 4 shows how to transform any SSMC-protocol to satisfy the simplified AC-framework based on certain techniques adopted in Pinkas' compiler. Section 5 provides an alternative transformation utilizing threshold homomorphic encryption. To demonstrate additional utilities

¹ We have only cited one early example of each.

of the AC-framework, Section 6 presents a game theoretic model using the framework to achieve Nash Equilibrium. Section 7 concludes the paper.

2. Related work

We first give a description and definitions of Secure Multiparty Computation; these are necessary to understand the rest of the paper. We then discuss previously proposed ideas that appear similar to our proposed AC-framework, and highlight the differences.

2.1. Secure multi-party computation

Yao first proposed the two-party comparison problem and developed a provably secure solution [6]. This was extended to multiparty computations by Goldreich et al. [4]. They developed a framework for secure multiparty computation and proved that computing a function privately is equivalent to computing it securely [10].

We start with the definitions for security in the semi-honest model. A semi-honest party (also referred to as honest but curious) follows the rules of the protocol using its correct input but is free to later use what it sees during execution of the protocol to compromise security. An informal definition of private two-party computation in the semi-honest model is: Let T_i be the input of party i , $\Pi_i(f)$ be the party i 's execution image of the protocol f and s_i be the result computed from f . f is secure if $\Pi_i(f)$ can be simulated from $\langle T_i, s_i \rangle$ and distribution of the simulated image is computationally indistinguishable from $\Pi_i(f)$. The above definition says that a computation is secure if the view of each party during the execution of the protocol can be effectively simulated given the input and the output of that party. This model guarantees that parties who correctly follow the protocol do not have to fear seeing data they are not supposed to. A formal definition can be found in [10].

The malicious model (guaranteeing that a malicious party cannot obtain private information from an honest one, among other things) adds considerable complexity due to the fact that the consistency of every step of execution with previous computations generally needs to be verified. While we do not give the full definition here, we note that there are three things the model cannot handle [10]: (1) parties refusing to participate in the protocol, (2) parties using other (valid) input in place of their actual data, and (3) parties aborting the protocol prematurely.

2.2. Other verification-based methods

Ideas proposed in [11,12] appear similar to what we have presented in this paper. However, both of them focus on the situation where verifications are mandatory and performed on the fly. This difference will become clear after we detail the AC-framework. Another key distinction is that our AC-framework can achieve a *practical* efficiency (in cases where there is no reason to suspect malicious behavior) not achievable by previous methods.

In addition, the framework presented in [11] adopts a game-theoretic approach in that participants are rational, and when periodically using an auditing device, the participants are expected to provide truthful information. The AC-framework proposed in this paper does not assume any such setting. On the other hand, the paper illustrates that the existence of such framework enables the design of a simple game theoretic model.

The general AC-framework was presented in [13], which presents AC-protocol to mine frequent itemsets. We introduced a preliminary version of the general AC-framework translation in [14]; the current work also presents an alternative transformation to construct an AC-protocol using threshold homomorphic encryption. Furthermore, this paper shows how to use the AC-framework to design a game theoretic model that achieves Nash Equilibrium.

3. The simplified AC-framework

We now present a simplified version of the AC-framework. Before presenting details, we clarify the following terminologies:

- *SSMC-protocol*: a protocol secure under the semi-honest model in the literature of secure multi-party computation (SMC);
- *MSMC-protocol*: a protocol secure under the malicious model under the context of SMC;
- *AC-protocol*: a protocol that satisfies the AC-framework.

In addition to the above terminologies, the terms honest and semi-honest are interchangeable for the rest of the paper, and the term “secure” means secure against an adversary computationally bounded in the traditional cryptographic sense (i.e., only capable of polynomial-time computations; unable to otherwise defeat the cryptographic schemes used in polynomial time).

The full AC-framework is proposed in [13]; here we provide its key definitions and simplified structures. Suppose Φ is a protocol satisfying all the requirements under the AC-framework. In general, the AC-framework provides means to examine if what participating parties have done during the execution of Φ is consistent with honest behaviors (expected under the semi-honest model). For instance, whether or not a party has followed the prescribed execution procedures of a protocol could be proved in the framework. The next two definitions define key components and conditions that a protocol needs to guarantee in the AC-framework.

Definition 1 (*Verifier*). A verifier is a third party (e.g., a court, a government agency, etc.) who oversees and validates the verification process of Φ .

Definition 2 (*AC-protocol*). An AC-protocol Φ must satisfy the following three requirements:

- (1) *Basic security*: Without consideration of the verification process, Φ satisfies the security requirements of an SSMC-protocol (an SMC-protocol secure under the semi-honest model).
- (2) *Basic structure*: The execution of Φ consists of two phases:
 - *Computation phase*: Compute the prescribed functionality and store information needed for the verification process.
 - *Verification phase*: An honest party (named as a prover hereafter) can succeed in proving that he or she has followed the protocol as prescribed.
- (3) *Sound verification*: Φ is sound providing that
 - the verification phase cannot be fabricated by a malicious party;
 - no additional information is leaked regarding the participating parties’ private inputs.

Note that unlike the computation phase, the verification phase stated in [Definition 2](#) is optional for each run of an AC-protocol. The verification process is performed merely when there is a need to verify whether or not the participating parties have behaved honestly or there is an accusation that a participating party did not behave honestly and thus may have obtained or disclosed information which should not be leaked. Since verification can detect any malicious behavior defined in the malicious model, it allows the accused party who correctly follows the protocol to prove it has not obtained information it should not have.

The ideal model for privacy is a trusted third party (TTP) [10] that learns all parties’ private data and computes the result. The goal of Secure Multiparty Computation is to achieve this without requiring such a trusted third party. Some privacy-preserving protocols utilize a third party to perform certain computations. The third party represents a party in whom all participating parties repose some degree of trust. In the AC-framework, the verifier is such a third party. It is needed only if verification is performed (e.g., serving the role of a mediator if there is a dispute), and does not learn anything regarding participating parties’ private inputs. Because the verification phase (used only when needed) under the AC-framework does not leak any additional information regarding the participating parties’ private inputs, and because the verification phase can be repeated as many times as needed with different verifiers, the degree of trust for a verifier is lower than for most third party protocols. In practice, it should be much easier to find a verifier under the AC-framework than for most third party protocols.

This simplified version of the AC-framework, especially the verification phase along with its soundness property, captures the most significant aspect of the AC-framework in that the verification phase is capable of detecting any malicious behaviors that can be prevented under the malicious adversary model, and at the same time, nothing regarding the private inputs of participating parties is leaked during the verification process.

The running time or complexity of an AC-protocol (verification phase) can be as or possibly even more inefficient than an MSMC-protocol; however, the computation phase of an AC-protocol should be more efficient because the verification phase of the protocol is not needed for every run. If the complexity of the computation phase of an AC-protocol were comparable to an MSMC-protocol, the MSMC-protocol would be sufficient and more effective than the AC-protocol for practical purposes. Therefore, a challenge in designing an AC-protocol is to ensure that the computation phase is efficient. In general, the implementation of an AC-protocol's computation phase is the same as that of an SSMC-protocol, except that additional information needs to be computed for the verification phase. Steps provided in the verification phase serve as guidance for a verifier, and the soundness of the verification phase needs to be proven.

4. SSMC to AC based on Pinkas' Compiler

For the rest of the paper, we use the term *AC-framework* to mean the simplified AC-framework introduced in Section 3. The goal of this section is to prove the following claim:

Claim 1. *There exists a generic two-party (excluding the verifier) AC-protocol that is nearly as efficient as the generic circuit-based SSMC-protocol.*

By *generic* protocols, we mean that protocols are represented as boolean circuits. It has been shown that for any polynomial-time algorithm, there exists a polynomial time generic secure protocol that achieves the same functionality [4,6]. Thus, to prove the above claim, we construct a generic AC-protocol from a circuit-based SSMC-protocol. Then we show that the generic AC-protocol provides security guarantees allowing detection of any behavior prevented by the corresponding generic MSMC-protocol. Next we briefly discuss the concept of two-party circuit-based secure function evaluation under the semi-honest model.

Algorithm 1: Circuit construction [15]

- 1: Randomly generate s-bit strings: $r_1, \bar{r}_1, \dots, r_t, \bar{r}_t, r_x, \bar{r}_x, r_y, \bar{r}_y$;
- 2: Randomly generate bijections: $\phi_1, \dots, \phi_r, \phi_x, \phi_y$ such that $\phi_i : \{r_i, \bar{r}_i\} \rightarrow \{0, 1\}$;
- 3: For each AND gate g_i , construct a set S_i as follows:
 - (a) Find the left and right input gates g_l and g_r of g_i ;
 - (b) Randomly generate s-bit strings: m_1, m_2, m_3, m_4 and compute $\bar{m}_1, \bar{m}_2, \bar{m}_3, \bar{m}_4$ as follows:

$$\begin{aligned} \bar{m}_1 &= m_1 \oplus \phi_i^{-1}(\phi_l(r_l) \wedge \phi_r(r_r)); & \bar{m}_2 &= m_2 \oplus \phi_i^{-1}(\phi_l(\bar{r}_l) \wedge \phi_r(r_r)) \\ \bar{m}_3 &= m_3 \oplus \phi_i^{-1}(\phi_l(r_l) \wedge \phi_r(\bar{r}_r)); & \bar{m}_4 &= m_4 \oplus \phi_i^{-1}(\phi_l(\bar{r}_l) \wedge \phi_r(\bar{r}_r)) \end{aligned}$$

- (c) Construct the pairs p_1, p_2, p_3 and p_4 as follows:

$$\begin{aligned} p_1 &= \langle E(m_1, r_l), E(\bar{m}_1, r_r) \rangle; & p_2 &= \langle E(m_2, \bar{r}_l), E(\bar{m}_2, r_r) \rangle \\ p_3 &= \langle E(m_3, r_l), E(\bar{m}_3, \bar{r}_r) \rangle; & p_4 &= \langle E(m_4, \bar{r}_l), E(\bar{m}_4, \bar{r}_r) \rangle \end{aligned}$$

- (d) $S_i \leftarrow \pi(\{p_1, p_2, p_3, p_4\})$, where π is a random permutation;

- 4: For each NOT gate g_i , construct the set S_i as follows:
 - (a) Find the input gate g_o of g_i ;
 - (b) $S_i \leftarrow \pi(\{E(\phi_i^{-1}(1 \oplus \phi_o(r_o)), r_o), E(\phi_i^{-1}(1 \oplus \phi_o(\bar{r}_o)), \bar{r}_o)\})$;
-

4.1. Secure circuit evaluation

Suppose Alice and Bob want to securely compute a function f based on their private inputs x and y . Let $f_A(x, y)$ and $f_B(x, y)$ denote the outputs of Alice and Bob, respectively. Without loss of generality, assume $f_A(x, y) = f_B(x, y)$, f consists of gates g_1, \dots, g_t and g_t is the output gate. Also, assume inputs x and y are single bit values represented by gates g_x and g_y .² Let $E(M, r)$ be a probabilistic encryption scheme where M is a message and r is a random value.

Algorithm 2: Secure function evaluation (SFE)

Require: x, y, f

I Bob:

- (1) Generate a combinatorial circuit (with AND and NOT logic gates) C_f to compute f ;
- (2) Scramble C_f based on Algorithm 1 and send Alice: C_f^g and $\phi_y^{-1}(y)$, where $C_f^g = \{S_1, \dots, S_t, \phi_t = \{\langle r_t, \phi_t(r_t) \rangle, \langle \bar{r}_t, \phi_t(\bar{r}_t) \rangle\}\}$;

II Alice:

- (1) Get $\phi_x^{-1}(x)$ from Bob via 1–2 oblivious transfer;
 - (2) Compute $f_A(x, y)$ and $f_B^g(x, y)$ from $\phi_x^{-1}(x)$, $\phi_y^{-1}(y)$ and C_f^g ;
 - (3) Send $f_B^g(x, y)$ to Bob, where $f_B^g(x, y)$ is the garbled value of Bob's output $f_B(x, y)$;
-

Following the description in [15], Algorithm 1 highlights key steps for constructing a scrambled circuit. The intuition is that one party (Bob) encrypts the boolean circuit gate by gate and the other party (Alice) evaluates the circuit. Since AND and NOT are sufficient for constructing any boolean circuit, Bob merely needs to know how to encrypt the two gates. Gates are encrypted using a set of random values corresponding to input or output values of every gate. To evaluate the encrypted circuit, Alice first obtains random values related to her and Bob's input values. Using these, she can decrypt one entry of every top level gate to get a new set of random values. Continuing this process, Alice will eventually obtain random values corresponding to the output. Using pre-defined mappings from Bob, Alice can obtain the actual result. More details can be found in [4,15,6]. Using the procedures presented in Algorithm 2, Alice and Bob can compute any function securely, and Algorithm 2 provides certain key steps. Note that step II(1) requires using an oblivious transfer (OT) protocol [16] to get the random value related to Alice's input.

Suppose f is the logical AND function: $\text{AND}: (x, y) \rightarrow x \wedge y$. Based on Algorithms 1 and 2, the next example shows how to compute it securely between Alice and Bob.

Example 1 (Refer to Fig. 1). Bob randomly generates s -bit strings: $r_1, \bar{r}_1, r_x, \bar{r}_x, r_y, \bar{r}_y$ and bijection mappings: $\phi_1 = \{\langle r_1, 1 \rangle, \langle \bar{r}_1, 0 \rangle\}$, $\phi_x = \{\langle r_x, 0 \rangle, \langle \bar{r}_x, 1 \rangle\}$ and $\phi_y = \{\langle r_y, 1 \rangle, \langle \bar{r}_y, 0 \rangle\}$. Then Bob randomly generates s -bit strings: m_1, m_2, m_3 and m_4 and compute the following:

$$\begin{aligned} \bar{m}_1 &= m_1 \oplus \phi_1^{-1}(\phi_x(r_x) \wedge \phi_y(r_y)) = m_1 \oplus \bar{r}_1 \\ \bar{m}_2 &= m_2 \oplus \phi_1^{-1}(\phi_x(\bar{r}_x) \wedge \phi_y(r_y)) = m_2 \oplus r_1 \\ \bar{m}_3 &= m_3 \oplus \phi_1^{-1}(\phi_x(r_x) \wedge \phi_y(\bar{r}_y)) = m_3 \oplus \bar{r}_1 \\ \bar{m}_4 &= m_4 \oplus \phi_1^{-1}(\phi_x(\bar{r}_x) \wedge \phi_y(\bar{r}_y)) = m_4 \oplus \bar{r}_1 \end{aligned}$$

² These assumptions only simplify and clarify our presentation, and the ideas, analyses and proofs presented next can be applied to any boolean circuit with multiple input and output gates.

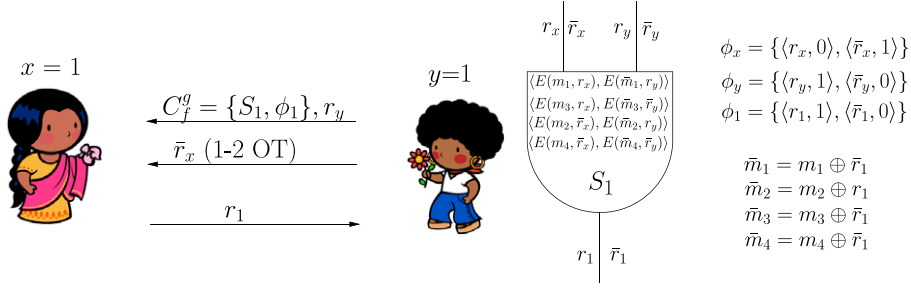


Fig. 1. The garbled circuit C_f^g is constructed by Bob, where r, \bar{r} and m are random s-bit strings, table entries of S_1 are randomly permuted, and $E(m,r)$ is a probabilistic encryption scheme.

Bob constructs four pairs:

$$p_1 = \langle E(m_1, r_x), E(\bar{m}_1, r_y) \rangle; \quad p_2 = \langle E(m_2, \bar{r}_x), E(\bar{m}_2, r_y) \rangle$$

$$p_3 = \langle E(m_3, r_x), E(\bar{m}_3, \bar{r}_y) \rangle; \quad p_4 = \langle E(m_4, \bar{r}_x), E(\bar{m}_4, \bar{r}_y) \rangle$$

Bob sets $S_1 = \pi(\{p_1, p_2, p_3, p_4\})$. Now suppose $x = 1$ and $y = 1$, then Bob sends S_1, r_y and ϕ_1 to Alice. Using 1–2 OT, Alice gets \bar{r}_x (since $\phi_x(\bar{r}_x) = 1$) from Bob. Based on \bar{r}_x and r_y , Alice can only decrypt one of the pairs in S_1 to get m_2 and \bar{m}_2 .³ Because $\bar{m}_2 \oplus m_2 = r_1$, from ϕ_1 , Alice can compute $f(x, y) = f(1, 1) = \phi_1(r_1) = 1$.

In the previous example, $f_A(x, y) = f_B(x, y) = 1$, so at the end, Alice simply sends 1 or r_1 to Bob.

4.2. From SSMC to AC

The circuit evaluation approach presented in Algorithm 1 and 2 is secure under the semi-honest model, but has vulnerabilities under the malicious adversary model. As stated in Section 2, the malicious model cannot handle situations where parties refuse to participate, input modification, or premature termination. Thus, we exclude these three situations for our analyses. However, the preceding protocol is only secure if we can guarantee that:

- (1) Bob correctly constructs the garbled circuit C_f^g of C_f ;
- (2) Alice sends the correct $f_B^g(x, y)$ value to Bob.

For the rest of this section, we show how to guarantee these two conditions under the AC-frame work. To start with, we first show how to meet the second condition by utilizing some key ideas proposed in [17] (used to compile an SSMC-protocol into an MSMC-protocol). The basic idea is that during the construction of C_f^g , Bob randomizes the output value. For the output wire ϕ_t , Bob chooses a random binary key $k \in \{0, 1\}$. If $k = 0$, the mapping ϕ_t is not changed; otherwise, the ordering of the entries in ϕ_t is reversed. Then, the modified ϕ_t is sent to Alice.

At the end of the evaluation of C_f^g , Alice knows the permuted 0 or 1 value of the output wire. Alice and Bob exchange values: Alice sends $f_B^g(x, y)$ to Bob and Bob sends k to Alice. This step should not be done directly as described because a party who gets the value first has an advantage by not sending his or

³ This example does not show how Alice knows which pair she can decrypt correctly, but this can be easily done in practice via message padding. For illustrative purpose, we omitted this step here.

her value to the other party. Therefore, this step should be performed via gradual release timed commitments [17,18]. Thereafter, for $b \in \{0, 1\}$, $c(b)$ denotes the gradual release timed commitment, and $c(r)$ denotes a regular commitment for a random chosen value r . Alice and Bob first exchange the commitments $c(b)$ and $c(k)$ of $f_B^g(x, y)$ and k . After that, Alice sends Bob the first hint message for opening $c(b)$. After verifying the hint is valid, Bob sends Alice the first hint for opening $c(k)$. Both parties continue to send and verify alternate hint messages for a number of rounds until they can open their received commitments. If one party aborts the protocol, say Bob, Alice can open the next commitment in polynomial time without Bob's hint. Although the protocol does not provide complete fairness, Bob only has a slight advantage. Plus, Alice is able to know, in polynomial time, the bit that Bob learned by cheating. (A construction of such gradual release timed commitments can be found in [17].)

The above steps do not guarantee that the commitments are related to the actual output of C_f . The correctness of Bob's commitment can be justified when the first condition is met. We will come back to this point later in this section. The correctness of Alice's commitment can be justified through a blind signature scheme [19,20] in that Bob can sign Alice's commitment without knowing its actual value. An example is Chaum's signature scheme: an RSA pair (N, e) , and secret key d . Let H be a hash function (e.g., SHA [21], MD5 [22]) whose range is in Z_N^* . The signature of a message M is defined as $(H(M))^d \bmod N$.

Alice constructs the commitment as follows: For the output wire of g_t , Alice generates commitments $c(0)$ and $c(1)$ for bit values 0 and 1, respectively. She chooses a random value $r \in_R Z_N^*$ and computes $c^h(0) = r^e \cdot H(c(0))$ and $c^h(1) = r^e \cdot H(c(1))$. Alice sends the two pairs $\langle c^h(0), 0 \rangle$ and $\langle c^h(1), 1 \rangle$ to Bob. Bob signs both commitments: $(c^h(0))^d = r \cdot (H(c(0)))^d$ and $(c^h(1))^d = r \cdot (H(c(1)))^d$ (all operations are modulo N). Based on the random permutation k and the mappings $\langle c^h(0), 0 \rangle$ and $\langle c^h(1), 1 \rangle$, Bob maps $r \cdot (H(c(0)))^d$ and $r \cdot (H(c(1)))^d$ to the 0 and 1 entries of the output table ϕ_t , respectively. He then sends Alice the scrambled circuit C_f^g along with the commitment $c(k)$ of k .

After the evaluation of C_f^g , Alice gets either $r \cdot (H(c(0)))^d$ or $r \cdot (H(c(1)))^d$. Suppose Alice gets $r \cdot (H(c(0)))^d$. She unblinds the signature to get $\langle c(0), (H(c(0)))^d \rangle$ and sends it to Bob. During the construction of C_f^g , since $c(0)$ and $c(1)$ are never disclosed, Bob does not know whether $c(0)$ is a 0-commitment or a 1-commitment when Bob receives $c(0)$ and $(H(c(0)))^d$ from Alice. On the other hand, Bob is able to verify if $c(0)$ corresponds to the correct output value by checking the signature $(H(c(0)))^d$. Thus, if Alice sends Bob a value other than $c(0)$, she is not able to compute a valid signature. As a consequence, Alice can be detected as being malicious. Finally, both parties gradually open their timed commitments to get the expected result.

The first condition, that C_f^g is constructed correctly, can be guaranteed during the verification phase under the AC-framework. During the verification phase, Alice sends C_f^g (received from Bob) to the verifier. Bob sends the verifier: $r_x, \bar{r}_x, r_y, \bar{r}_y, \langle c^h(0), 0 \rangle$ and $\langle c^h(1), 1 \rangle$ (received from Alice). From these values, the verifier can decrypt C_f^g and examine whether C_f^g correctly computes the function f .

Based on the tools described previously, we are ready to define a secure function evaluation protocol under the AC-framework (AC-SFE). Keep in mind that the AC-SFE protocol can detect any malicious behavior defined under the malicious model, if verification is performed. Key steps of the AC-SFE protocol are presented in Algorithms 3 and 4 regarding the computation phase and verification phase, respectively. According to previous discussions, Algorithm 3 is straightforward. The computation phase of AC-SFE adopts a secure signature scheme denoted by *Sign* (e.g., RSA [23]) which is used during the verification process.⁴ In order to fit the domain of a digital signature scheme, we also use a hash function (e.g., SHA [21], MD5 [22]) to compute the hash value of an intended message and then sign the hash value instead. This is implicit when a signature is computed in Algorithm 3.

⁴ For clarity, we assume an authenticated channel between Alice and Bob and show only signatures needed for verification; in practice all messages would be signed to ensure integrity.

Algorithm 3 (AC-SFE: Computation phase)

Require $x, y, f, \text{Sign}, H, \parallel$, where Sign is a secure signature scheme, H is a hash function and \parallel is a message concatenation operator

I Alice:

- (1) Compute $\langle c^h(b), b \rangle$, where $c^h(b) = r^e \cdot H(c(b)) \bmod N$, for $b \in \{0, 1\}$;
- (2) Send Bob: $I_A = \{\langle c^h(0), 0 \rangle, \langle c^h(1), 1 \rangle\}$ and $\text{Sign}_A(I_A)$;

II Bob:

- (1) **Abort**, if I_A is not consistent with $\text{Sign}_A(I_A)$;
- (2) Generate a circuit (with AND and NOT logic gates) C_f to compute f ;
- (3) Scramble C_f based on Algorithm 1 and additionally do the following: randomly select $k \in \{0, 1\}$, sign on $c^h(0)$ and $c^h(1)$ and generate $I_B = \{c(r_x), c(\bar{r}_x), c(r_y), c(\bar{r}_y)\}$;
- (4) Based on k , set $\phi_i = \{\langle r_i, r \cdot (H(c(b)))^d \rangle, \langle \bar{r}_i, r \cdot (H(c(\bar{b})))^d \rangle\}$;
- (5) Send Alice: $I_B, C_f^g, \text{Sign}_B(I_B \parallel C_f^g)$, and $\phi_y^{-1}(y)$, where $C_f^g = \{S_1, \dots, S_t, \phi_i, c(k)\}$;

III Alice:

- (1) **Abort**, if I_B and C_f^g are not consistent with $\text{Sign}_B(I_B \parallel C_f^g)$;
- (2) Get $\phi_x^{-1}(x)$ from Bob via 1–2 oblivious transfer;
- (3) **Abort**, if $\phi_x^{-1}(x)$ and $\phi_y^{-1}(y)$ are not consistent with commitments in I_B ;
- (4) Without loss of generality, assume $r \cdot (H(c(b)))^d$ is computed from $\phi_x^{-1}(x)$, $\phi_y^{-1}(y)$ and C_f^g ;
- (5) **Abort**, if $c(b)$ is not consistent with $r \cdot (H(c(b)))^d$;
- (6) Unblind $r \cdot (H(c(b)))^d$ and send Bob: $\langle c(b), (H(c(b)))^d \rangle$ and $\text{Sign}_A(c(b) \parallel (H(c(b)))^d)$;

IV Bob: **Abort**, if $c(b)$ is not consistent with $(H(c(b)))^d$ or if $\langle c(b), (H(c(b)))^d \rangle$ is not consistent with $\text{Sign}_A(c(b) \parallel (H(c(b)))^d)$;

V Alice & Bob: Gradually open the timed commitments: Alice $\leftarrow k$ and Bob $\leftarrow b$;

The next example shows how AC-SFE is executed in practice. We follow the same settings and notations as in Example 1 (where f is a logic AND function). For illustration purpose, we assume every message passed between Alice and Bob is valid, so certain values (e.g., commitments in I_B) and some steps (e.g., steps II (1), III (1, 3, 5) and IV of Algorithm 3) are implicit or omitted in the following example.

Algorithm 4 (AC-SFE: verification phase)

- 1: Alice sends the verifier: I_B, C_f^g and $\text{Sign}_B(I_B \parallel C_f^g)$;
- 2: Bob sends the verifier: $I_A, \text{Sign}_A(I_A)$ and $\{r_x, \bar{r}_x, r_y, \bar{r}_y\}$; {The verifier can catch malicious parties from executing the following procedures sequentially:}
- 3: Alice behaved maliciously, if I_B and C_f^g are not consistent with $\text{Sign}_B(I_B \parallel C_f^g)$;
- 4: Bob behaved maliciously, if $\langle c^h(0), 0 \rangle$ and $\langle c^h(1), 1 \rangle$ are not consistent with $\text{Sign}_A(\langle c^h(0), 0 \rangle \parallel \langle c^h(1), 1 \rangle)$;
- 5: Alice behaved maliciously, if $c^h(0)$ is not a commitment of 0 or $c^h(1)$ is not a commitment of 1;
- 6: Bob behaved maliciously, if:
 - (a) $\{r_x, \bar{r}_x, r_y, \bar{r}_y\}$ is not consistent with I_B ;
 - (b) the correctness of C_f^g 's construction cannot be verified from $\langle c^h(0), 0 \rangle, \langle c^h(1), 1 \rangle$ and $\{r_x, \bar{r}_x, r_y, \bar{r}_y\}$;

Example 2 (Refer to Fig. 2). Bob randomly generates s -bit strings: $r_1, \bar{r}_1, r_x, \bar{r}_x, r_y, \bar{r}_y$ and mappings: $\phi_x = \{\langle r_x, 0 \rangle, \langle \bar{r}_x, 1 \rangle\}$ and $\phi_y = \{\langle r_y, 1 \rangle, \langle \bar{r}_y, 0 \rangle\}$. He then randomly selects the value $k \in \{0, 1\}$. Assume $k = 1$ in this example. Based on the values $\langle c^h(0), 0 \rangle$ and $\langle c^h(1), 1 \rangle$ received from Alice, Bob computes $\phi_1 = \{\langle r_1, r \cdot (H(c(0)))^d \rangle, \langle \bar{r}_1, r \cdot (H(c(1)))^d \rangle\}$. Bob constructs $S_1 = \pi\{p_1, p_2, p_3, p_4\}$ the same way as in Example 1. We also assume $x = 1$ and $y = 1$, then Bob sends S_1, r_y, ϕ_1 and $c(k)$ (commitment of k) to Alice. Using 1–2 OT, Alice gets \bar{r}_x (since $\phi_x(\bar{r}_x) = 1$) from Bob. Based on \bar{r}_x and r_y , Alice can decrypt one of the pair in S_1 to get m_2 and \bar{m}_2 . Because $\bar{m}_2 \oplus m_2 = r_1$, from ϕ_1 , Alice gets $r \cdot (H(c(0)))^d$ and unblinds it to get $(H(c(0)))^d$. According to the $c(0)$ value she generated before, Alice is able to know $(H(c(0)))^d$ is Bob's signature

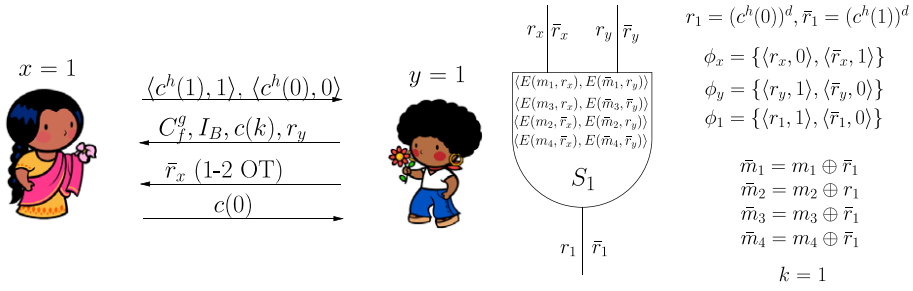


Fig. 2. k is a random bit indicating if the mappings in ϕ_1 of the output table S_1 are swapped; $I_B = \{c(r_x), c(\bar{r}_x), c(r_y), c(\bar{r}_y)\}$ is the set of input commitments; gradually open $c(k)$ and $c(0)$ to get the actual output: $0 \oplus k = 1$.

related to $c(0)$. She then sends $c(0)$ to Bob. At the end, they gradually open the commitments: Alice gets the value k and Bob gets the value 0 (the committed value of $c(0)$). Alice can compute $f_A(x, y) = 0 \oplus k = 1$, and Bob can compute $f_B(x, y) = 0 \oplus k = 1$.

Note that as stated previously, $k = 1$ implies the ordering in the output table ϕ_t is swapped; therefore, the output computed from C_f^g (e.g., 0) XOR with k produces the actual output.

4.2.1. Security and soundness of AC-SFE

The key difference between SSMC-SFE and AC-SFE's computation phase is that the messages, passed between Alice and Bob in AC-SFE, contain additional commitments. Due to the (computationally or perfect) hiding property of the commitment scheme, these commitments do not convey any more information than these messages themselves. Therefore, the security analysis of AC-SFE is the same as that of SSMC-SFE. Next, we show that the verification phase of AC-SFE is sound.

Claim 2. *The verification phase of AC-SFE is sound (Definition 2) provided that its computation phase did not terminate prematurely and Alice and Bob did not collude.*

Proof. Two conditions need to be examined: whether Alice sent Bob the correct commitments, and whether Bob constructed G_f^g correctly. (Fig. 3 summarizes messages sent to the verifier.) To verify these two conditions, the verifier initially needs to be certain about the authenticity of the garbled circuit C_f^g and information used to construct the circuit. Steps 3–4 of Algorithm 4 achieve this requirement with reasons as follows: since we assume that the computation phase of AC-SFE did not abort, Alice did receive the correct message and signature pair: I_B, C_f^g and $Sign_B(I_B || C_f^g)$. Thus, if the inconsistency is detected at step 3, the verifier can be certain that Alice did not send the expected value, and consequently, she is malicious. The same reasoning applies at step 4. If no malicious behavior detected at steps 3–4, the verifier can conclude that he or she has the authenticated information to verify the two conditions.

Now the first condition can be verified if $c^h(0)$ and $c^h(1)$ are correct commitments of 0 and 1 . That neither is a correct commitment signals that Alice did not follow the protocol to compute the expected commitments. Step 6 of Algorithm 4 examines the construction of C_f^g . Step 6(a) checks whether or not Bob sent the verifier the values to scramble the circuit. After $\{r_x, \bar{r}_x, r_y, \bar{r}_y\}$ is verified, these values can be used to decrypt the C_f^g . Then, it is easy to see if C_f^g computes f correctly. \square



Fig. 3. Verification phase.

We need to emphasize that the verification phase of AC-SFE does not disclose any information regarding the private inputs x and y to the verifier because the verifier only has values used to construct C_f^g but not values (e.g., $c(b)$) evaluated from C_f^g . Hence, we relinquish the responsibility for checking consistencies at steps III (3, 5) and IV of [Algorithm 2](#) to the users. In addition, note that for the verification phase, we did not consider the situation where the protocol terminates prematurely. The aborting of a protocol is a very complex issue because many factors could be involved. It would be very interesting to design a verification process that handles such situations.

4.3. AC vs. SSMC

The computation phase of AC-SFE ([Algorithm 3](#)) is not much different from SFE under the semi-honest model (SSMC-SFE, [Algorithm 2](#)). The AC-SFE protocol requires computing additional commitments for the input wires and the output wire. In general, these computations are very simple comparing to the circuit construction and circuit evaluation phases. Therefore, the complexity of AC-SFE's computation phase is linearly bounded by the number of gates in C_f which is the same as SSMC-SFE.

Since the verification phase of AC-SFE ([Algorithm 4](#)) is optional, efficiency of the verification phase is not an important issue (see [Section 3](#) for more discussion). Because the main cost of the verification phase is to decrypt every garbled gate, its complexity is also linearly bounded by the number of gates in C_f . Therefore, both computation phase and verification phase of AC-SFE are virtually as efficient as SSMC-SFE under the assumption that the number of the output gates is much smaller than that of C_f^g .

On the other hand, AC-SFE is more powerful than SSMC-SFE since the verification phase of AC-SFE can detect any malicious behavior defined under the malicious model. Therefore, the AC-SFE protocol provides an example that an AC-protocol is more useful and practical than an SSMC-protocol and yet almost without any efficiency loss.

4.4. AC vs. MSMC

Given an SSMC-protocol, the compiler proposed in [\[17\]](#) can be utilized to transform it into a secure protocol under the malicious model (MSMC-SFE). The basic steps involve: Bob creates $2l$ copies of C_f^g and sends them to Alice. Alice randomly chooses l out of $2l$ copies to verify if all l circuits compute f correctly. If they do, Alice needs to evaluate the rest l circuits. The reason to generate $2l$ garbled circuits is to bound the error probability (where Bob was malicious but did not detect) by $(\frac{3}{4})^{\frac{2l}{2}}$. There are other mechanisms adopted to bound Alice's error probability. Generally speaking, AC-SFE is $2l$ times more efficient than (this version of) MSMC-SFE, and yet the verification phase of AC-SFE is able to detect any malicious behaviors that can be prevented in MSMC-SFE.

It has been shown that for any polynomial-time algorithm, there exists a polynomial time secure protocol that achieves the same functionality, and generic solutions are presented in [\[4,6\]](#). Since our implementation of AC-SFE is based on a generic protocol under the semi-honest model, the construction of the AC-SFE protocol validates [Claim 1](#).

5. SSMC to AC based on homomorphic encryption

In this section and under the random oracle model [\[24\]](#), we present an alternative construction based on homomorphic encryption to prove [Claim 1](#). We first provide a brief description of threshold homomorphic cryptosystem. Based on that, we give an overview of how Alice and Bob can securely compute any two party function $f(x, y)$ represented as a boolean circuit C_f in semi-honest model. At the end of the section, we show the transformation from SSMC to AC. Note that under the context of this section, the term SSMC (MSMC or AC)-SFE actually means SSMC (MSMC or AC)-SFE whose implementation relies on threshold homomorphic encryption.

5.1. Threshold homomorphic cryptosystems

Let $E: R \times X \rightarrow Y$ be a probabilistic public key encryption scheme, where R , X and Y are finite domains identified with an initial subset of integers and $D: Y \rightarrow X$ be a private decryption algorithm, such that $\forall (r, x) \in R \times X$, $D(E(r, x)) = x$. Furthermore, the scheme has the following properties:

- The encryption function is injective with respect to the second parameter, i.e., $\forall (r_1, x_1), (r_2, x_2) \in R \times X$, $E(r_1, x_1) = E(r_2, x_2) \Rightarrow x_1 = x_2$.
- The encryption function is additive homomorphic, i.e., $\forall (r_1, x_1), (r_2, x_2) \in R \times X$, $E(r_1, x_1) +_h E(r_2, x_2) = E(r_3, x_1 + x_2)$, where r_3 can be computed from r_1, r_2, x_1 and x_2 in polynomial time.
- The encryption function has semantic security as defined in [25]. Informally speaking, a set of ciphertexts do not provide additional information about the plaintext to an adversary with polynomial-bounded computing power.
- The domain and the range of the encryption system is suitable.

Paillier's public key encryption scheme [26] has these properties. Honest behaviors in the malicious model can be enforced using efficient zero-knowledge (ZK) proofs [27] under the random oracle model [24]. In order to utilize ZK proof techniques, the Paillier's scheme has following additional properties (two-party case) [28]:

- *Threshold decryption*: Given a public–private key pair (e.g., k_{pu} and k_{pr}), the private key k_{pr} consists of two shares (k_{pr}^1, k_{pr}^2) privately distributed between two parties;
- Given a message M , $c = E(r, M)$ (k_{pu} is implicit in E) can be efficiently computed along with its ZK proof. For instance, suppose c is computed by party Alice, then Alice can also efficiently generate ZK proof, denoted as $ZK(c)$, such that without disclosing M , Alice is able to prove to a verifier that c is the correct encryption of some value Alice has;
- $M_j = D_{k_{pr}^j}(c)$ ($j \in \{1, 2\}$) can be efficiently computed along with its ZK proof, denoted as $ZK(M_j)$, such that $ZK(M_j)$ can be used to show the decryption is performed correctly;
- M_1 and M_2 can be combined to produce M , but either M_1 or M_2 alone does not leak any information regarding M .
- Given a constant k and a ciphertext $E(r, M)$, $k \times_h E(r, M) = E(r', k \cdot M)$ can be efficiently computed along with its ZK proof.

For the rest of the paper, we adopt $E(M)$ and $D(c)$ to represent $E(r, M)$ and $D_{k_{pr}^j}(c)$ wherever the context is clear.

5.2. Secure function evaluation using threshold homomorphic encryption

In the following discussion, we follow the protocols given in [29]. Also, we assume that Paillier's scheme is used as a threshold homomorphic encryption scheme and C_f is a combinatorial circuit (with AND and NOT logic gates) to compute a function f .

Before Alice and Bob start evaluating the circuit C_f , Alice encrypts her i th input bit x_i and sends every encrypted bit $E(x_i)$ to Bob. Bob encrypts his i th input bit y_i and sends every encrypted bit $E(y_i)$ to Alice. Note that Alice and Bob cannot decrypt each other's encrypted inputs because they just have a share of the secret key. Now given the encrypted inputs to the circuit C_f , they need to calculate the corresponding AND and NOT gates on encrypted input bits without revealing anything until they calculate the last output gates.

Given encrypted bit $E(b)$, both Alice and Bob can calculate the encrypted result of a NOT gate $E(-b)$ as $E(-b) = E(1) +_h (-1 \times_h E(b))$ using the homomorphic encryption properties. The challenge is to calculate the encrypted output of the AND gate given its encrypted input bits $E(b_1)$ and $E(b_2)$. In other words, given inputs $E(b_1)$ and $E(b_2)$, how can Alice and Bob calculate $E(b_1 \wedge b_2)$? This can be achieved using the homomorphic encryption properties. Key steps are highlighted in Algorithm 5. At step 1, Alice and Bob choose two random

numbers in Z_N which are used to hide b_1 when decryption is performed at step 3. At the end, utilizing algebraic and homomorphic properties, Alice and Bob can independently calculate $E(b_1 \wedge b_2)$ (since $a_1 \cdot b_2 + a_2 \cdot b_2 = -b_1 \cdot b_2$).

Algorithm 5 (SFE of AND using homomorphic encryption)

Require $E(b_1)$, $E(b_2)$ and N (the public key as in Paillier’s system)

- 1: **Alice:** choose $d_1 \in {}_R Z_N$, compute $E(d_1)$ and **send** it to Bob
 - Bob:** choose $d_2 \in {}_R Z_N$, compute $E(d_2)$ and **send** it to Alice
 - 2: **Alice:** compute $e = E(b_1) + {}_h E(d_1) + {}_h E(d_2)$, $e_a = D_{k_1}^{-1}(e)$ and **send** e_a to Bob
 - Bob:** compute $e = E(b_1) + {}_h E(d_1) + {}_h E(d_2)$, $e_b = D_{k_2}^{-1}(e)$ and **send** e_b to Alice
 - 3: **Alice:** compute $a = D_{k_1}^{-1}(e_b) = b_1 + d_1 + d_2$ and $a_1 = a - d_1$
 - Bob:** compute $a = D_{k_2}^{-1}(e_a) = b_1 + d_1 + d_2$ and $a_2 = -d_2$
 - 4: **Alice:** calculate $E(a_1 \cdot b_2) = a_1 \times {}_h E(b_2)$ and **send** it to Bob
 - Bob:** calculate $E(a_2 \cdot b_2) = a_2 \times {}_h E(b_2)$ and **send** it to Bob
 - 5: **Alice:** compute $E(b_1 \wedge b_2) = E(a_1 \cdot b_2) + {}_h E(a_2 \cdot b_2)$
 - Bob:** compute $E(b_1 \wedge b_2) = E(a_1 \cdot b_2) + {}_h E(a_2 \cdot b_2)$
-

The procedures described above enable Alice and Bob to calculate the encrypted output of any intermediate gate given the encrypted input values. At the final stage, the threshold decryption property allows Alice and Bob to jointly decrypt the final output of the circuit C_f .

5.3. From SSMC to AC using ZK proofs

Here, we briefly discuss how to construct an AC-SFE based on Algorithm 5 using ZK proofs. The computation phase of the AC-SFE is the same as the SSMC-SFE (constructed based on threshold homomorphic encryption). As stated in Section 4, we assume an authenticated channel between Alice and Bob (all messages would be signed to ensure integrity in practice) and the computation phase of the AC-SFE does not terminate prematurely.

Under these assumptions, implementation of the verification phase is straightforward. To ensure honest behaviors during the evaluation of C_f , the verifier needs to check: (1) each party’s inputs are encrypted correctly and (2) each AND gate is evaluated correctly. To check the first condition, Alice and Bob generate ZK proofs corresponding to the encrypted input values, and send the verifier the encrypted input values and their ZK proofs. If any ZK proof cannot be verified, the cheating is detected. To ensure the second condition, each party needs to generate ZK proofs (for every AND gate) regarding values computed at steps 1, 2 and 4 of Algorithm 5. For instance, Fig. 4 shows how a verifier can check if $E(b_2)$ is computed correctly by Bob at step 1. As shown in the figure, during the verification phase, Bob needs to compute the ZK proof of $E(d_2)$ ($ZK(E(d_2))$) and sends $ZK(E(d_2))$ to the verifier. Alice sends $E(d_2)$ and $Sign_B(E(d_2))$ to the verifier. If $E(d_2)$ and $Sign_B(E(d_2))$ are not consistent, the verifier can claim that Alice was malicious during the computation phase because we assume that the computation phase did not abort (i.e., messages received by both parties are valid) and without aborting, it is expected that Alice received valid message and signature pairs. After that, the verifier can use $ZK(E(d_2))$ to check if $E(d_2)$ was computed correctly during the computation phase by Bob. Utilizing these procedures, the verifier can verify the behaviors of both Alice and Bob. In addition, the soundness (Definition 2) of the verification phase is guaranteed by the properties of ZK proof.

The MSMC-SFE (presented in [29]) requires computation and verification of ZK-proofs regarding the encrypted input values and values for evaluating every AND gates. Thus, the AC-SFE’s computation phase



Fig. 4. Verification phase based on ZK proofs.

is more efficient than the MSMC-SFE since all the expensive computation and verification of ZK proofs are delayed to the verification phase. In addition, the complexity of the verification phase is bounded by the number of computation and verification of ZK proofs which is polynomially bounded by the input sizes. Thus, the computation complexity of the verification phase is no worse than that of the MSMC-SFE. An example of ZK-based AC-protocol can be found in [13], and its empirical complexity analysis matches our theoretic analysis here.

We have presented two constructions to prove Claim 1: the first approach is based on Pinkas' compiler and the second approach is based on threshold homomorphic encryption. The first approach requires limited interactions between Alice and Bob. Although the second approach requires interaction for every evaluation of an AND gate, it does provide a simpler implementation. In addition, since ZK proofs are independent from one to another, if a certain degree of error probability is allowed, the verification phase under the second approach could be more efficient by just verifying a random sample among all ZK proofs. We will formally analyze the tradeoff between an error bound and a selected sample size in the future.

6. A game theoretic model utilizing AC

So far, we have shown the existence of a general AC-protocol. In addition to detecting malicious behaviors, the AC-framework can be utilized to design a game theoretic model to answer the following question: how often should either Alice or Bob verify whether or not the other party has behaved honestly during the computation phase of an AC-protocol? To answer this question, we will use tools from game theory. Following the general assumption in game theory [30], we assume that Alice and Bob both are rational. In other words, they will cheat if and only if cheating is profitable for them.

For simplicity, we model the case where only Alice wants to verify whether Bob has cheated or not. The same analysis given here could directly be applied for the case where Bob wants to verify as well. We assume that if Alice wants to verify and if Bob did not cheat, Alice pays for the entire verification costs plus compensation to Bob (since Alice initiates the verification process). Let c be the compensation paid by Alice to Bob, and v denote Alice's total cost of verification. If Alice does not verify the circuit construction and Bob has cheated, she loses l due to the inaccurate result. When Bob cheats and is not caught, he gains g . If Bob is caught cheating, he needs to pay a penalty of g_p , and Alice gains g_v . If there is no cheating and no verification, we assume that Bob gains f_b and Alice gains f_a from the correct function result. Fig. 5 shows the relative gains corresponding to different actions. V and NV , respectively, correspond to the cases where Alice verifies and does not verify. C and NC , respectively, correspond to the cases where Bob cheats and does not cheat during the computation phase of an AC-protocol.

To analyze the above two party strategic game, we need some basic definitions from game theory. Let p (resp. $(1 - p)$) be the probability that Alice verifies, i.e., executes the verification procedures of the AC-protocol, (resp. does not verify) and q (resp. $1 - q$) be the probability that Bob cheats (resp. does not cheat) during the computation phase. Furthermore, we can define the expected utilities for Alice (u_A) and Bob (u_B) given p , q and Fig. 5:

$$u_A(q, p) = p \cdot q \cdot g_v + (1 - p) \cdot q \cdot -l + (1 - q) \cdot p \cdot -v + (1 - q) \cdot (1 - p) \cdot f_a$$

$$u_B(q, p) = p \cdot q \cdot -g_p + (1 - p) \cdot q \cdot g + (1 - q) \cdot p \cdot c + (1 - q) \cdot (1 - p) \cdot f_b$$

Using the above utility functions and notations, we next define Nash Equilibrium.

Definition 3 (*Nash Equilibrium* [30]). A strategy profile $\sigma^* = (q^*, p^*)$ is a mixed strategy Nash Equilibrium in the verification game with utility functions u_A , u_B if the following inequalities hold:

		Alice	
		V	NV
Bob	C	$-g_p, g_v$	$g, -l$
	NC	$c, -v$	f_b, f_a

Fig. 5. Verification game.

$$u_A(q^*, p^*) \geq u_A(q^*, p) \quad (1)$$

$$u_B(q^*, p^*) \geq u_B(q, p^*) \quad (2)$$

where $0 \leq p, p^*, q, q^* \leq 1$.

Intuitively, the above definition states that if all participants predict that a particular equilibrium will occur, then no player has an incentive to deviate from the equilibrium strategy. Using the Nash Equilibrium definition, we can prove the following theorem:

Theorem 4. Given $v, l, c, g_p, g_v, g, f_b, f_a > 0$, the game defined in Fig. 5 has a mixed strategy Nash Equilibrium $p^* = 0$ and $q^* = 0$ if $f_b \geq g$ and has a equilibrium $p^* = \frac{g-f_b}{c+g_p+g-f_b}$ and $q^* = \frac{f_a+v}{g_v+l+f_a+v}$ if $f_b < g$.

Proof. First, let us consider the case where $f_b \geq g$ and show that $p^* = 0$ and $q^* = 0$ satisfies the inequalities 1 and 2. Given $q^* = 0$, we can show that p^* is the best strategy as follows:

$$u_A(0, p) = p \cdot -v + (1-p) \cdot f_a = f_a - p \cdot (v + f_a)$$

Clearly $u_A(0, p)$ is a decreasing function in p . Therefore, $p = 0$ maximizes $u_A(0, p)$. Therefore, inequality 1 is satisfied.

Similarly, given $p^* = 0$, and $f_b \geq g$, we can show that $q^* = 0$ is the best strategy. Note that if $p^* = 0$, we can rewrite the $u_B(q, 0)$ as follows:

$$u_B(q, 0) = q \cdot g + (1-q) \cdot f_b = f_b + q \cdot (g - f_b)$$

Since $g - f_b \leq 0$ when $f_b \geq g$, $q = 0$ maximizes $u_B(q, 0)$. Therefore, we can conclude that inequality 2 is satisfied.

Now, let us consider the case where $g > f_b$. Again, given the p^*, q^* , we just need to show that inequalities 1 and 2 are satisfied. Let us first prove that given q^* , we cannot find any other p that gives higher utility than p^* . We first calculate $u_A(q^*, p)$:

$$\begin{aligned} u_A(q^*, p) &= p \cdot q^* \cdot g_v + (1-p) \cdot q^* \cdot -l + (1-q^*) \cdot p \cdot -v + (1-q^*) \cdot (1-p) \cdot f_a \\ &= p \cdot q^* \cdot g_v + q^* \cdot -l + p \cdot q^* \cdot l + p \cdot -v + p \cdot q^* \cdot v + f_a - p \cdot f_a - f_a \cdot q^* + p \cdot q^* \cdot f_a \\ &= p[q^* \cdot (g_v + l + v + f_a) - v - f_a] + f_a - q^* \cdot (l + f_a) \\ &= p \left[\frac{f_a + v}{g_v + l + f_a + v} \cdot (g_v + l + v + f_a) - v - f_a \right] + f_a - q^* \cdot (l + f_a) = f_a - q^* \cdot (l + f_a) \end{aligned}$$

The above equation implies that given q^* , the utility for Alice is the same for all p values. Therefore, p^* satisfies inequality 1. Similarly, let us prove that given p^* , we cannot find any other q that gives higher utility than q^* . As before, we can calculate $u_B(q, p^*)$:

$$\begin{aligned} u_B(q, p^*) &= p^* \cdot q \cdot -g_p + (1-p^*) \cdot q \cdot g + (1-q) \cdot p^* \cdot c + (1-q) \cdot (1-p^*) \cdot f_b \\ &= p^* \cdot q \cdot -g_p + q \cdot g - p^* \cdot q \cdot g + p^* \cdot c - q \cdot p^* \cdot c + f_b - f_b \cdot p^* - f_b \cdot q + f_b \cdot p^* \cdot q \\ &= q \cdot [-p^* \cdot (g_p + g + c - f_b) + g - f_b] + p^* \cdot (c - f_b) + f_b \\ &= q \cdot \left[-1 \cdot \frac{g - f_b}{c + g_p + g - f_b} \cdot (g_p + g + c - f_b) + g - f_b \right] + p^* \cdot (c - f_b) + f_b = p^* \cdot (c - f_b) + f_b \end{aligned}$$

Again given p^* , all q values will yield the same utilities for Bob. This implies that q^* satisfies the inequality 2. \square

Theorem 4 indicates two different Nash Equilibria. The first equilibrium corresponds to case where Bob has no incentive to cheat (i.e., $f_b \geq g$) and correspondingly Alice does not need to verify at all. In other words, if we can guarantee that other party cannot gain more by cheating, we do not need to verify at all.

In the second case (i.e., $f_b < g$) we have a Nash Equilibrium where Alice verifies with probability $p^* = \frac{g-f_b}{c+g_p+g-f_b}$ and Bob cheats with probability $q^* = \frac{f_a+v}{g_v+l+f_a+v}$. Since $p^* = \frac{1}{1+\frac{g_p+c}{g-f_b}}$, when $\frac{g_p+c}{g-f_b}$ increases, the probability of verification at the equilibrium point decreases. Similarly, since $q^* = \frac{1}{1+\frac{g_v+l}{f_a+v}}$, when $\frac{g_v+l}{f_a+v}$ increases,

the cheating probability at the equilibrium point decreases. Based on these information, Alice and Bob could decide on the amount of penalties and gains that maximize their profits when they execute an AC-protocol in practice.

7. Conclusion

Confidentiality is an extremely important issue in privacy-preserving data mining (PPDM). SMC-techniques are among basic tools in designing PPDM applications. An SSMC-protocol, if followed, prevents information disclosure. However, it may be possible for a dishonest party to undetectably cause disclosure by not following the protocol correctly. At the other end of the spectrum, a protocol secure under the malicious model definitely erases these security concerns. Nevertheless, efficient MSMC-protocols appear to be difficult to design. Fortunately, the AC-framework has opened up the opportunity for more efficient protocols and maintain strong security against malicious adversaries.

In this paper and through the generic construction of two AC-SFE protocols, we have shown that for any polynomial-time algorithm, there exists a polynomial-time AC-protocol that achieves the same functionality and is capable of detecting any malicious behaviors that can be prevented under the malicious adversary model. AC-SFE is virtually as efficient as SSMC-SFE, and at the same time, since its verification phase can detect any malicious behavior, it has substantial practical advantages over SSMC-SFE. In addition, the verification phase of AC-SFE does not disclose any information regarding each party's private input, and AC-SFE is more efficient than its counterpart under the malicious model. In addition, to demonstrate the utility of the AC-framework, we have designed a simple game theoretic model that achieves Nash Equilibrium.

The game theoretic setting also allows us to determine under what settings the AC-framework is better than the malicious model. Informally, if the gain from cheating g is high relative to the penalty g_p and gain from honest behavior f_b , then the need for verification rises. As the (amortized) cost of verification approaches the cost of running the malicious protocol, the malicious protocol becomes more attractive.

A related avenue for exploration is *partial verification*. The AC-framework provides the flexibility of checking only some gates or ZK-proofs; the malicious model requires that every gate or ZK-proof be verified. However, some gates or ZK-proofs may be more critical to achieving a gain from cheating; it is possible that randomly selecting a subset of gates to verify (with a non-uniform probability of each gate) could give a better amortized cost/benefit than the analysis of Section 6.

Although the verification phase of AC-SFE is optional, it does provide incentives for a party to behave honestly (as shown in the game theoretic model) because once the verification is executed, any malicious behavior can be detected. Due to the privacy-preserving property of the verification phase, it is easy to choose the verifier in practice. Even though the AC-SFE protocol does not explicitly state the penalty related to the detection of malicious behaviors in the verification phase, in real life, this can be addressed through contract signing. Before using the protocol, both parties should agree on the penalty when any malicious behavior is detected.

In theory, we have showed that the two generic constructions of AC-SFE's computation phase are more efficient than their counterparts in the malicious model. In the future, we would like to analyze extensively and precisely how efficient the two constructions of AC-SFE can be in practical terms comparing to both SSMC-SFE and MSMC-SFE.

References

- [1] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the Protection of Individuals with regard to the Processing of Personal Data and on the Free Movement of such Data, Official Journal of the European Communities No I. (281) (1995) 31–50. URL <http://ec.europa.eu/justice_home/fsj/privacy/law/index_en.htm>.
- [2] Standard for Privacy of Individually Identifiable Health Information, Federal Register 67 (157) (2002) 53181–53273. URL <<http://www.hhs.gov/ocr/hipaa/finalreg.html>>.
- [3] J.M. Perry, Statement of John M. Perry, President and CEO, Cardsystems Solutions, Inc. before the United States House of Representatives Subcommittee on Oversight and Investigations of the Committee on Financial Services. <<http://financialservices.house.gov/hearings.asp?formmode=detail&hearing=407&comm=4>> (Jul 21 2005), <<http://financialservices.house.gov/hearings.asp?formmode=detail&hearing=407&comm=4>>.

- [4] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game – a completeness theorem for protocols with honest majority, in: 19th ACM Symposium on the Theory of Computing, 1987, pp. 218–229. URL <http://doi.acm.org/10.1145/28395.28420>.
- [5] A.C. Yao, Protocols for secure computation, in: Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, IEEE, 1982, pp. 160–164.
- [6] A.C. Yao, How to generate and exchange secrets, in: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, IEEE, 1986, pp. 162–167.
- [7] M.J. Atallah, M. Bykova, J. Li, M. Karahan, Private collaborative forecasting and benchmarking, in: Proceedings of the Second ACM Workshop on Privacy in the Electronic Society (WPES), Washington, DC, 2004.
- [8] Y. Lindell, B. Pinkas, Privacy preserving data mining, *Journal of Cryptology* 15 (3) (2002) 177–206. <http://www.research.ibm.com/people/l/lindell/id3_abs.html>.
- [9] M. Naor, B. Pinkas, R. Sumner, Privacy preserving auctions and mechanism design, in: Proceedings of the First ACM Conference on Electronic Commerce, ACM Press, 1999.
- [10] O. Goldreich, *The Foundations of Cryptography*, vol. 2, Cambridge University Press, 2004. URL <<http://www.wisdom.weizmann.ac.il/oded/PSBookFrag/prot.ps>> (Chapter: General Cryptographic Protocols).
- [11] R. Agrawal, E. Terzi, On honesty in sovereign information sharing, in: Proceedings of the 10th International Conference on Extending Database Technology (EDBT), 2006.
- [12] R. Gennaro, M.O. Rabin, T. RabinG, Simplified vss and fast-track multiparty computations with applications to threshold cryptography, in: Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing, 1998, pp. 101–111.
- [13] W. Jiang, C. Clifton, AC-framework for privacy-preserving collaboration, in: SIAM International Conference on Data Mining, Minnesota, Minneapolis, 2007.
- [14] W. Jiang, C. Clifton, Transforming semi-honest protocols to ensure accountability, in: Workshop on Privacy Aspects of Data Mining (PADM06) in conjunction with the Sixth IEEE International Conference on Data Mining (ICDM06), Hong Kong, China, 2006.
- [15] M. Franklin, M. Yung, Varieties of secure distributed computing, in: Proceedings of Sequences II, Methods in Communications, Security and Computer Science, Positano, Italy, 1991, pp. 392–417. <<http://www.cs.ucdavis.edu/franklin/pubs/survey.ps>>.
- [16] M. Naor, B. Pinkas, Efficient oblivious transfer protocols, in: Proceedings of SODA 2001 (SIAM Symposium on Discrete Algorithms), Washington, DC, 2001.
- [17] B. Pinkas, Fair secure two-party computation, in: Proceedings of IACR Eurocrypt (EUROCRYPT03), Warsaw, Poland, 2003, pp. 87–105.
- [18] D. Boneh, M. Naor, Timed commitments, in: *Advances in Cryptology – Crypto 2000*, LNCS, vol. 1880, Springer-Verlag, 2000, pp. 236–254.
- [19] D. Chaum, Blind signatures for untraceable payments, in: *Advances in Cryptology – Crypto 1982*, 1982, pp. 199–203.
- [20] M. Bellare, C. Namprempre, D. Pioncheval, M. Semanko, The power of rsa inversion oracles and the security of chaum’s rsa-based blind signature scheme, in: *Proceedings of Financial Cryptography 2001*, 2001.
- [21] Secure hash standard, Tech. Rep. FIPS PUB 180-1, National Institutes of Standards and Technology, Apr. 17 1995. URL <<http://www.itl.nist.gov/fipspubs/fip180-1.htm>>.
- [22] R.L. Rivest, The md5 message-digest algorithm, Technical Report RFC 1321, Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc. (Apr. 1992).
- [23] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* 21 (2) (1978) 120–126. <http://doi.acm.org/10.1145/359340.359342>.
- [24] M. Bellare, P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, in: *ACM Conference on Computer and Communications Security*, 1993, pp. 62–73.
- [25] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof systems, in: Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC’85), Providence, Rhode Island, USA, 1985, pp. 291–304.
- [26] P. Paillier, Public key cryptosystems based on composite degree residuosity classes, in: *Advances in Cryptology – Eurocrypt ’99 Proceedings*, LNCS, vol. 1592, Springer-Verlag, 1999, pp. 223–238.
- [27] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems, *Journal of ACM* 38 (1991) 690–728.
- [28] I. Damgard, M. Jurik, J. Nielsen, A generalization of paillier’s public-key system with applications to electronic voting., 2003.
- [29] R. Cramer, I. Damgård, J.B. Nielsen, Multiparty computation from threshold homomorphic encryption, in: *EUROCRYPT*, 2001, pp. 280–299.
- [30] M.J. Osborne, A. Rubinstein, *A Course in Game Theory*, MIT Press, 1999.



Wei Jiang is a Ph.D. candidate for computer science at Purdue University. He has a MS from Purdue University and BS from The University of Iowa. His research interests include privacy-preserving data mining, data integration and privacy issues in federated search environment.



Chris Clifton is an Associate Professor of Computer Science at Purdue University. He has a Ph.D. from Princeton University, and Bachelor's and Master's degrees from the Massachusetts Institute of Technology. Prior to joining Purdue in 2001, Chris had served as a Principal Scientist at The MITRE Corporation and as an Assistant Professor of Computer Science at Northwestern University. His research interests include privacy, data mining, data security, database support for text, and heterogeneous databases.



Murat Kantarcioglu is currently an assistant professor of computer science at University of Texas at Dallas. He had a Ph.D. degree from Purdue University in 2005. He received his master's in Computer Science from Purdue University in 2002 and his bachelor degree in computer engineering from METU, Ankara, Turkey in 2000. During his graduate years, he worked as a summer intern at IBM Almaden Research Center and at NEC Labs. His research interests lie at the intersection of Privacy, Security, Data Mining and Databases: Security and Privacy issues raised by data mining; Distributed Data Mining techniques; Security issues in Databases; Applied Cryptography and Secure Multi-Party Computation techniques; Use of data mining for intrusion and fraud detection.