2006

AC-Framework for Privacy-Preserving Collaboration

Wei Jiang

Chris Clifton
Purdue University, clifton@cs.purdue.edu

Report Number:
06-015

AC-FRAMEWORK FOR PRIVACY-PRESERVING COLLABORATION

Wei Jiang
Chris Clifton

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #06-015
August 2006

AC-Framework for Privacy-Preserving Collaboration

Wei Jiang
Dept. of Computer Science,
Purdue University, W. Lafayette, IN
wjiang@cs.purdue.edu

Chris Clifton
Dept. of Computer Science,
Purdue University, W. Lafayette, IN
clifton@cs.purdue.edu

Abstract

The secure multi-party computation (SMC) model provides means for balancing the use and confidentiality of distributed data. 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 most applications. In this paper, we propose a novel framework: accountable computing (AC) framework, which is sufficient or practical for many applications without the complexity and cost of a SMC-protocol under the malicious model. Furthermore, to show the applicability of the AC-framework, we present an application under this framework regarding privacy-preserving mining frequent itemsets.

1 Introduction

Privacy and security, particularly maintaining confidentiality of data, have become a challenging issue with advances in information and communication technology. The ability to communicate and share data has many benefits. The idea of an omniscient data source carries great value to research and data dissemination. Witnessing the cost of duplicated medical tests or the damage from errors resulting from incomplete or incorrect information, such a data source could substantially reduce waste and inefficiency.

On the other hand, an omniscient data source eases misuse, such as the growing problem of identity theft. To prevent misuse of data, there is a recent surge in laws mandating protection of confidential data, such as the European Community privacy standards [5], U.S. healthcare laws [11], 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. 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 data sharing without having to actually 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) [8, 19, 20] has recently emerged as an answer to this problem. Informally, if a protocol meets the SMC definitions, the participating parties learn mere the final result and whatever can be inferred from the final result and their own inputs. A simple example is Yao's millionaire problem [19]: 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 [3], data analysis [12] and auctions [14].[1]

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 [8]. 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.

A SMC-protocol secure under the semi-honest model (or a SSMC-protocol) rarely provides sufficient security for practical applications. For example, two competing transportation companies want to know if they can collaborate to achieve better efficiency and consequently reduce operational costs. Assume there exists a SSMC-protocol that searches for possible overlap of the two companies' trucking routes. It is difficult to convince the companies to utilize the protocol because it is unacceptable to assume that two competing companies trust each other to follow the protocol. On the other hand, if one company can guarantee the other company that it has behaved honestly during each execution step of the protocol, then a collaboration between the two parties becomes possible. A SMC-protocol secure under the malicious model (or a MSMC-protocol) generally provides such a guarantee, but the complexity of a MSMC-protocol commonly prevents it from being adopted in practice.

Imagine another scenario: multiple parties *authorized* to see each other's data want to compute a shared result; nevertheless, open disclosure of the data to non-participating parties is prohibited. In order to avoid the cost or liability from disclosing the data, every party finds that its best interest is to follow a protocol secure under the semi-honest model. What happens if data is disclosed? Clearly it is the fault of the original owner of the data provided that other parties followed the protocol. On the other hand, since a party may have behaved dishonestly, the owner can accuse others and claim that liability should be shared. At this point, the other party would like to prove that they did follow the protocol and consequently show that they could not have seen the data (unless the owner had behaved dishonestly). The scenario leads us to a new framework: accountable computing (AC).

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 someone else must have improperly disclosed data. This provides substantial practical utility over a semi-honest protocol. In addition, although a malicious adversary participating in an AC-protocol may learn things that they should not 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. 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 introduce and analyze the AC-framework as well as to demonstrate

[1] We have only cited one early example of each.

its practicality. Section 2 presents current state of the art from the literature of secure multiparty computation. Section 3 introduces the framework and provides guidelines for designing an AC-protocol. Section 4 illustrates the feasibility and applicability of the framework in the field of privacy-preserving data mining. Finally, section 5 concludes the paper.

2 Related Work / Background

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

2.1 Secure Multi-party Computation

Yao first postulated the two-party comparison problem (Yao's Millionaire Protocol) and developed a provably secure solution [20]. This was extended to multiparty computations by Goldreich et al. [8]. They developed a framework for secure multiparty computation, and in [9] proved that computing a function privately is equivalent to computing it securely.

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.

A formal definition of private two-party computation in the semi-honest model is given below.

Definition 1 *Let* $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ *be a functionality, and* $f_1(x,y)$ (resp., $f_2(x,y)$) *denote the first* (resp., second) *element of* $f(x,y)$. *Let* Π *be two-party protocol for computing* f. *The* view *of the first* (resp., second) *party during an execution of* Π *on* (x,y), *denoted* $\text{VIEW}_1^{\Pi}(x,y)$ (resp., $\text{VIEW}_2^{\Pi}(x,y)$), *is* $(x,r,m_1,\ldots,m_t)$ (resp., $(y,r,m_1,\ldots,m_t)$), *where* r *represents the outcome of the first* (resp., second) *party's internal coin tosses, and* m_i *represents the* i^{th} *message it has received. The* OUTPUT *of the first* (resp., second) *party during an execution of* Π *on* (x,y), *denoted* $\text{OUTPUT}_1^{\Pi}(x,y)$ (resp., $\text{OUTPUT}_2^{\Pi}(x,y)$) *is implicit in the party's own view of the execution, and* $\text{OUTPUT}^{\Pi}(x,y) = \left(\text{OUTPUT}_1^{\Pi}(x,y), \text{OUTPUT}_2^{\Pi}(x,y)\right)$.

(general case) *We say that* Π privately computes f *if there exist probabilistic polynomial-time algorithms, denoted* S_1 *and* S_2, *such that*

$$\{(S_1(x, f_1(x,y)), f(x,y))\}_{x,y} \overset{C}{\equiv} \left\{\left(\text{VIEW}_1^{\Pi}(x,y), \text{OUTPUT}^{\Pi}(x,y)\right)\right\}_{x,y}$$
$$\{(S_2(y, f_2(x,y)), f(x,y))\}_{x,y} \overset{C}{\equiv} \left\{\left(\text{VIEW}_2^{\Pi}(x,y), \text{OUTPUT}^{\Pi}(x,y)\right)\right\}_{x,y}$$

where $\overset{C}{\equiv}$ *denotes computational indistinguishability by (non-uniform) families of polynomial-size circuits.*

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.

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 [9]:

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.

While many of the existing practical SMC-style protocols do provide guarantees beyond that of the semi-honest model (such as guaranteeing that individual data items are not disclosed to a malicious party), few meet all the requirements of the malicious model.

2.2 Other Verification-based Methods

Ideas proposed in [2, 6] 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 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 [2] adopts a game-theoretic approach in that participates are rational, and using an auditing device periodically are expected to provide truthful information.

3 The AC-Framework

In this section, we introduce the accountable computing (AC) framework. Before presenting details, we first 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: represents a protocol secure in the proposed AC-framework.

In addition to the above terminologies, the terms *honest* and *semi-honest* are interchangeable for the rest of the paper.

Suppose Φ is a protocol satisfying all the requirements under the AC-framework. In general, the AC-framework provides a participating party means to prove what it has 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 model. For the rest of the chapter, we first present essential definitions related to the AC-framework. We conclude the section by differentiating the AC-framework from the SMC model.

Definition 2 (Accountable Behavior) *Given a protocol Φ under the AC-framework, an accountable behavior $\Gamma_{\alpha \times \beta}$ specifies how participating parties should behave, and it has two related components α, β:*

- ***Verifier*** $\alpha \in \{$*Participating-party, Third-party*$\}$*: an entity who oversees and validates the verification process, where participating-party indicates the validation of the verification process is supervised by a participating party or parties, and third-party indicates the validation is supervised by a third party (i.e., a court, a government agency, etc).*

- ***Degree of disclosure*** β*: specifies what information can be disclosed during the verification process.*

According to the above definition, if α = Participating-party, $\Gamma_{\alpha \times \beta}$ can be interpreted as: an expected (or accountable) behavior can be verified among participating parties, and the information disclosed during the verification process must be consistent with β.

For practical purpose, we have classified the verifiers into two categories: participating-party and third-party. The AC-framework allows a participating party to be the verifier so that a well established or reputable party has the opportunity to evaluate if its first-time collaborator is trustworthy. On the other hand, a relatively unknown party can have the chance to prove its credibility to the other collaborating party under the proviso that the other party can be trusted based on its well-known image or other reputations. Thus, the verification process could serve as a mechanism in building trust among participating parties and reduce costs in establishing well-purposed collaboration. In case there is a dispute among participating parties, the verification process must be conducted under the supervision of a third entity so that any malicious party can be held accountable. In addition, a random verification can be performed as a spot check to audit the integrity of participating parties.

Degree of disclosure only applies to the verification process. The benefits of proving innocence may outweigh privacy concerns, or only take place in a trusted environment (e.g., a courtroom.) This component allows participating parties to decide what is more important. If the verifier is trusted, such as a court, disclosure of private information to the court may not be a problem during the verification process. Also, what can be disclosed during the verification process must be agreed on by all relevant participating parties before the execution of any AC-protocol.

Next we define conditions that a protocol needs to guarantee in the AC-framework.

Definition 3 (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 a SSMC-protocol (a 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 (we name such a party as a prover thereafter) can succeed in verifying an accountable behavior.*

3. ***Sound Verification****:* Φ *is sound providing that the verification phase cannot be fabricated by a malicious party.*

Note that unlike the computation phase, the verification phase stated in Definition 3 is optional for each run of an AC-protocol. Details follow later in the section.

3.1 General Assumptions

In this sub-section, we clarify two key assumptions adopted throughout the framework: one is related to the nature of involving entities, and the other concerns the number of malicious parties allowed.

3.1.1 Nature of Involving Entities

In the AC-framework, the involving entities consist of both the verifier and participating parties, and nature means a party behaves either semi-honestly or maliciously under the context of this paper.

Nature of Verifier: In general, we assume that there is one or a group of verifier(s) and that the verifier always behaves honestly during the verification process. It would not make sense to have a malicious verifier because if the malicious verifier does not want to be convinced, no one can succeed in the verification process.

Nature of Participating Parties: If a participating party is the verifier, we assume it is always an honest entity. Other participating parties can be either semi-honest or malicious.

3.1.2 Bounds on the Number of Malicious Parties

For certain models, SMC-protocols only exist when majority of participating parties are honest, and some situations require a majority of $\frac{2}{3}$. For this paper, we merely consider two-party protocols under the AC framework, and we require at least one of the two participating parties is honest (or semi-honest).

3.2 AC-framework vs. SMC

Here we lay out the essential differences between the AC-framework and the SMC model. In general, AC-protocols should be compared with SSMC and MSMC protocols on the following criteria: basic structure, security definitions and computation complexity.

3.2.1 Basic Structure

According to Definition 3, the verification phase along with the accountable behavior are the key features that distinguish the AC-framework from the SMC model. A protocol that satisfies the AC-framework needs to compute not only the correct results but also additional information needed for a party to verify the accountable behavior.

In addition, the verification phase is optional for each execution of an AC-protocol. The verification process is performed merely when there is an accusation that a participating party did not behave honestly regarding the accountable behavior and thus may have obtained or disclosed information which should not be leaked if it behaved semi-honestly. Verification allows the accused party who is in fact semi-honest to prove its innocence.

3.2.2 Security Definitions

Under the SMC model, the security definitions are generally based on two types of adversaries: semi-honest and malicious. On the contrary, the two adversary models are no longer relevant at least from the stand point of a verifier since we assume a verifier always behave semi-honestly as stated in section 3.1. Also, we do not enforce any constraint on the behaviors of participating parties, and they can be either semi-honest or malicious as long as they are not verifiers. Furthermore, it is inaccurate to assume a party whose actions need to be proved against the accountable behavior is semi-honest. Therefore, security definitions under the AC-framework do not distinguish between a semi-honest adversary and a malicious adversary.

The AC-framework is more flexible from its security point of view due to the Γ component stated in Definition 2. Since the accountable behavior can be anything, the AC-framework is generally more applicable than the SMC model.

3.2.3 Computation Complexity

Regardless various accountable behaviors, the computation complexity under the AC-framework is classified into two phases: computation phase and verification phase. Computation phase is required for each run of the protocol, and it produces the expected result and all necessary information needed for verification phase. Verification process could be optional for each run unless there is a dispute or an expected (or accountable) behavior needs to be verified.

The running time or complexity of an AC-protocol can be as (or possibly even more) inefficient as a 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 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.

3.3 Guideline for Designing AC-Protocol

Based on Definition 3, we outline basic procedures that can be used to implement a protocol under the AC-framework. The key procedures highlighted in Figure 1 consist of three parts: behavior specification, computation phase and verification phase.

Behavior specification defines what an expected or accountable behavior is and its related components. The implementation of the computation phase is the same as that of a 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. It needs to be proven that a malicious participating party cannot convince the verifier that it behaved honestly during the computation phase of the protocol.

According to the definitions and the guideline presented in this section, we proceed to present a case study that shows how an actual data mining task can be implemented under the AC-framework.

1: Behavior Specification:

- Define a verifiable behavior:
- Specify its related components.

2: Computation Phase:

- Design a protocol Φ secure under the semi-honest model;
- Compute all necessary information to support a sound verification process, and modify Φ accordingly.

3: Verification Phase:

- State verification procedures including who should do what during the verification process;
- Prove that the verification procedures along with additional information computed in the computation phase constitute a sound verification process under Definition 3.

Figure 1: Guideline for Designing an AC-protocol

4 Case Study: Finding Frequent Itemsets (FFI)

Several algorithms have been developed for privacy-preserving association rule mining using the secure multiparty computation framework. Most are based on the Apriori algorithm [1]; the key step is to securely compute frequency of a set of candidate itemsets without disclosing each party's private dataset. One approach, suggested in [7, 21], is to use a secure dot product protocol. In this section, to demonstrate the usefulness and applicability of the AC-framework, we first introduce a secure two-party protocol to compute frequent itemsets under the semi-honest model proposed in [7, 21]. We then show how to modify such protocol into one that satisfies the AC-framework, allowing the semi-honest behavior to be verified.

4.1 FFI under the semi-honest model (SSMC-FFI)

The protocols presented in [7, 21] are basically a secure dot product protocol between two vectors whose entries are either 0 or 1 values. Let $\vec{v}_1, \vec{v}_2$ be two vectors with size m of parties P1 and P2 respectively, and $\vec{v}_j[i]$ denotes the i^{th} bit of $\vec{v}_j$. Formally, define $\text{FFI}(\vec{v}_1, \vec{v}_2) \rightarrow \delta$, where $\delta = \vec{v}_1 \bullet \vec{v}_2$.

Let $E : R \times X \rightarrow Y$ be a probabilistic public key encryption scheme, such as those proposed in [4, 13, 16], 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., $\forall(r_1, x_1), (r_2, x_2) \in R \times X$, $\prod(E(r_1, x_2), E(r_2, x_2)) = E(r_3, x_1 + x_2)$, where r_3 can be computed from r_1, r_2, x_1 and x_2

in polynomial time. ($\prod$ is the function to "add" two encrypted values; multiplication in the systems listed above.)

- The encryption function has semantic security as defined in [10]. 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.

Key steps of the FFI protocol are highlighted in Algorithm 1. P1, at step 1, encrypts individual

Algorithm 1 FFI Protocol

Require: $\vec{v}_1, \vec{v}_2, E$, where $|\vec{v}_1| = |\vec{v}_2| = m$ and E has certified public key parameters

1: P1:

(a). Encrypt $\vec{v}_1$: $x_i \leftarrow E(r, \vec{v}_1[i])$, for $i = 1, \ldots, m$;
r is randomly chosen for each $\vec{v}_1[i]$;

(b). Send $x_1, \ldots, x_m$ to P2.

2: P2:

(a). Compute $\hat{\delta} \leftarrow \prod_{\forall i \wedge \vec{v}_2[i]=1} x_i$;

(b). Send $\hat{\delta}$ to P1.

3: P1:

(a). Compute $\delta \leftarrow D(\hat{\delta})$

(b). Send δ to P2.

value in its private vector $\vec{v}_1$ and sends them to P2. At step 2(a), the symbol $\prod$ indicates that the encrypted $\vec{v}_1[i]$ values are combined to produce the encrypted dot product value denoted by $\hat{\delta}$; the common characteristics among all these values is that their corresponding $\vec{v}_2[i]$ values must be 1. At step 3(a), P1 computes the actual dot product value δ, and P1 sends δ to P2 at the end.

Refer to table 1 where a dataset is vertically partitioned between P1 and P2, and it can be reconstructed via one-to-one join on the global identifier attribute Tr#. Assume P1 and P2 want to know if $\{abc\}$ is a frequent itemset. P1 first creates the vector $\vec{v}_1 = \vec{a} \wedge \vec{b} = (1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1) \wedge (0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 1) = (0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1)$, where $\vec{a}, \vec{b}$ are the column vectors related to attribute a, b of P1's dataset and $\wedge$ indicates the logic AND operator. P2 creates $\vec{v}_2 = \vec{c} = (1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1)$. After applying the FFI protocol, both parties get $\vec{v}_1 \bullet \vec{v}_2 = 3$ without disclosing the private vectors $\vec{v}_1$ and $\vec{v}_2$ to each other. If the minimum support is 2, then $\{abc\}$ is one of the frequent itemsets. Details regarding the correctness and security analyses of the FFI protocol can be found in [7, 21].

4.2 Attacks on SSMC-FFI

Since the FFI protocol presented in Algorithm 1 is secure under the semi-honest adversary model, it has two vulnerabilities under the malicious adversary model:

Table 1: P1 and P2 's Data (left and right respectively)

Tr#	a	b
1	1	0
2	1	0
3	0	0
4	0	1
5	1	1
6	1	1
7	0	0
8	1	1
9	1	1

Tr#	c	d	e
1	1	0	1
2	1	0	0
3	0	0	0
4	0	1	1
5	1	1	1
6	0	1	0
7	0	0	0
8	1	1	1
9	1	1	1

1. Input modification after the FFI protocol is initiated:

2. P1 sends P2 incorrect final result.

Although under the malicious model, input modification cannot be prevented, the input should not be modified further once the protocol is initiated. Any intentionally miscalculations at step 1 and step 2 are related to this issue. Therefore, under the AC-framework, any inconsistencies at these two steps should be detected during the verification phase.

In addition, even if P1 receives the right $\hat{\delta}$ value at step 3(b), P1 can send P2 any value instead of the actual δ value. As a result, this malicious behavior should be detected under the framework whenever it happens. Next section shows how to change the SSMC-FFI protocol into an AC-FFI protocol whose verification phase is capable of detecting the two malicious behaviors stated above if they have ever occurred during the computation phase.

4.3 FFI under the AC-Framework (AC-FFI)

Before introducing the AC-FFI protocol, we first define the accountable behavior $\Gamma_{\alpha\times\beta}$ associated with AC-FFI:

- Prevent input modification after the initiation of SSMC-FFI protocol;
- Prevent P1 from sending an incorrect result to P2 at the final step;
- α = Third-party;
- β: a random permutation of each private input vector and the final result of AC-FFI can be disclosed to the third party verifier.

Note that $\Gamma_{\alpha\times\beta}$ implies the prevention of any malicious behavior in the computation phase of AC-FFI. Thus, the verification phase of AC-FFI needs to verify every step in the computation phase.

Key steps of AC-FFI's computation phase are provided in Algorithm 2. The protocol adopts a secure signature scheme denoted by *Sign* (e.g., RSA [17]) which is used during the verification process. In order to fit the domain of a digital signature scheme, we also use a hash function (e.g., SHA [15], MD5 [18]) to compute the hash value of an intended message and then sign the hash

value instead. We also assume that both parties agree on a random permutation π to prevent the verifier V in Algorithm 3 from seeing the private inputs of both parties. Other parts of the computation phase are identical to SSMC-FFI.

Algorithm 2 AC-FFI Protocol - Computation phase

Require: $\vec{v}_1, \vec{v}_2, E, Sign, H$, where $|\vec{v}_1| = |\vec{v}_2| = m$, E has certified public key parameters, $Sign$ is a secure signature scheme and H is a hash function

1: P1: $\vec{v}_1' \leftarrow \pi(\vec{v}_1)$, send $[H(\vec{v}_1'), Sign_{P1}(H(\vec{v}_1'))]$ to P2;

2: P2:

(a). **Abort** if $H(\vec{v}_1')$ and $Sign_{P1}(H(\vec{v}_1'))$ cannot be verified;

(b). $\vec{v}_2' \leftarrow \pi(\vec{v}_2)$, send $[H(\vec{v}_2'), Sign_{P2}(H(\vec{v}_2'))]$ to P1;

3: P1:

(a). **Abort** if $H(\vec{v}_2')$ and $Sign_{P2}(H(\vec{v}_2'))$ cannot be verified;

(b). Encrypt $\vec{v}_1'$: $x_i \leftarrow E(r, \vec{v}_1'[i])$, for $i = 1, \ldots, m$;
r is randomly chosen for each $\vec{v}_1'[i]$;

(c). Set $X = x_1, \ldots, x_m$ and send $[X, Sign_{P1}(H(X))]$ to P2.

4: P2:

(a). **Abort** if X and $Sign_{P1}(H(X))$ cannot be verified

(b). Compute $\hat{\delta} \leftarrow \prod_{\forall i \wedge \vec{v}_2'[i]=1} x_i$;

(c). Send $[\hat{\delta}, Sign_{P2}(H(\hat{\delta}))]$ to P1.

5: P1:

(a). **Abort** if $\hat{\delta}$ and $Sign_{P2}(H(\hat{\delta}))$ cannot be verified

(b). Compute $\delta \leftarrow D(\hat{\delta})$

(c). Send $[\delta, Sign_{P1}(H(\delta))]$ to P2.

6: P2: **Abort** if δ and $Sign_{P1}(H(\delta))$ cannot be verified

The verification phase of AC-FFI is presented in algorithm 3. The verification phase is basically a reconstruction of what actually occurred during the computation phase by using the permuted inputs and messages sent during the computation phase. Steps 1 and 2 in algorithm 3 require P1 and P2 to send the messages they received during the computation phase.

At step 3, the verifier examines (via (a), (c) and (e)) if every message P1 sent is valid and examines (via (b) and (d)) if P1 correctly performed every required computation. Because P1 is required to send three messages and to perform two computations, the tasks conducted by the verifier at step 3 is complete in validating P1's behavior. The implementation of step 4 follows the same reasoning in validating P2's behavior, and consequently, we say that the verification phase is

Algorithm 3 AC-FFI protocol: Verification phase

Require: H, E, where H and E are used in the computation phase

1: P1: send $I_{P1} = [\vec{v}_1', Sign_{P2}(H(\vec{v}_2')), \hat{\delta}, Sign_{P2}(H(\hat{\delta}))]$ and D (decryption key) to the verifier V:

2: P2: send $I_{P2} = [\vec{v}_2', Sign_{P1}(H(\vec{v}_1')), X, Sign_{P1}(H(X)), \delta, Sign_{P1}(H(\delta))]$ to V;

3: When P2 is honest, V can catch P1 cheating if:

(a). $\vec{v}_1'$ and $Sign_{P1}(H(\vec{v}_1'))$ are not consistent;

(b). $D(X) \neq \vec{v}_1'$;

(c). X and $Sign_{P1}(H(X))$ are not consistent;

(d). $D(\hat{\delta}) \neq \delta$;

(e). δ and $Sign_{P1}(H(\delta))$ are not consistent.

4: When P1 is honest, V can catch P2 cheating if:

(a). $\vec{v}_2'$ and $Sign_{P2}(H(\vec{v}_2'))$ are not consistent;

(b). $D(\hat{\delta}) \neq \vec{v}_1' \bullet \vec{v}_2'$;

(c). $\hat{\delta}$ and $Sign_{P2}(H(\hat{\delta}))$ are not consistent.

complete regarding $\Gamma_{\alpha\times\beta}$.

4.3.1 Security and Soundness of AC-FFI

The key difference between SSMC-FFI and AC-FFI's computation phase is that the messages P1 and P2 receive in AC-FFI contain additional signatures, but the signatures do not convey any more information than the messages themselves. Therefore, the security analysis of AC-FFI is the same as that of SSMC-FFI, refer to [7, 21] for more details. Next, we show that the verification phase of AC-FFI is sound.

Claim 1 *The verification phase of AC-FFI is sound (definition 3) provided that one of the two parties is honest and the verification phase is complete in verifying the accountable behavior* $\Gamma_{\alpha\times\beta}$ *associated with AC-FFI.*

PROOF. Since we have showed that the verification phase is complete, to prove this claim, we only need to show any malicious behavior can be detected as long as one party is honest. First, suppose P2 is honest. Then the verifier is certain that I_{P2} is legitimate. Based on the three signatures contained in I_{P2}, the verifier can determine (through steps 3(a),3(c) and 3(e) in algorithm 3) the validity of $\vec{v}_1'$ in I_{P1} and two messages that P1 sent at steps 3(c) and 5(c) in the computation phase (algorithm 2).

Once the verifier confirms $\vec{v}_1'$ in I_{P1} is legitimate, the verifier (using the decryption key D at step 3(c) in algorithm 3) can determine if X was computed correctly at step 3(b) in the computation phase. If anything is inconsistent, either D is the incorrect key or P1 did not encrypt $\vec{v}_1'$ properly. Either way, the verifier can catch P1 cheating. After this, the verifier can confirm whether or not

D is valid, and consequently, the verifier can confirm the validity of the calculation at step 5(b) in algorithm 2.

The above analyses show a complete reconstruction of what P1 actually did in the computation phase. Since P2 is honest, any inconsistency in the reconstruction process leads to the fact that P2 did behave maliciously during the computation phase. Therefore, P1 cannot mislead the verifier in the verification phase as long as P2 is honest. On the other hand, if P1 is honest, based on the same reasoning, P2 cannot mislead the verifier either. We can conclude that the verification phase of AC-FFI is sound. □

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.

4.3.2 Advantages of AC-FFI

So far, we have provided a real-life application that shows the promise of the AC-framework. Comparing to the SSMC-FFI protocol, the computation phase of the AC-FFI protocol additionally needs to compute 5 hash values and 5 digital signatures, but these calculations are negligible. The hash cost is based on the size of the vector, but it is a very small cost compared to the cost of homomorphic encryption (which must also be applied to each item in the vector). The cost of a signature is comparable to the cost of an individual encryption because it is applied to the hash value whose size is constant. Thus the total cost of the protocol remains dominated by the cost of homomorphically encrypting the vectors, and it is not appreciably greater than that of SSMC-FFI. In addition, because of a sound verification phase, the AC-FFI protocol inherits significant advantage over the SSMC-FFI protocol in enforcing honest behaviors.

The AC-FFI protocol further confirms our intuition: the computation phase of an AC-protocol can be as efficient as a SSMC-protocol. Furthermore, since malicious behavior can be detected by the verification process, the verification phase provides participating parties the incentive to follow the protocol correctly; in other words, participating parties are more likely to output the correct result under the AC-framework than does a SMC-protocol secure under the semi-honest model.

5 Conclusion / Future Work

Confidentiality is an extremely important issue in data security. Even when different data holders are allowed to see each other's data, they may not choose to do so when they collaborate to achieve a common goal because they do not want to get accused that they are the ones to disclose certain confidential information. In this paper, we present the *accountable computing* (AC) framework that allows an honest party to prove innocence to a third independent entity when it has followed a protocol correctly. Such a framework has much potential in practice since it provides more incentives for a party to behave honestly.

A 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 malicious protocols appear to be difficult to design. The AC-framework provides a party verifiability, and its general structure allows possible design of more

efficient protocols than the malicious model because the verification process does not need to be carried out during the execution of an AC-protocol.

Under the AC-framework, we do not explicitly state the penalty related to the detection of malicious behaviors in the verification phase. In practice, this can be addressed through contract signing, and before using any AC-protocol, both parties should agree on the penalty should malicious behavior be detected.

In the paper, we also presented a secure finding frequent itemsets protocol that meets the definitions of the AC-frame- work. The protocol is nearly as efficient as its counterpart in the semi-honest model. An honest participating party can verify its honesty to a third entity. The disclosure in the verification process is very limited to the fact that the verifier only sees the permuted inputs and all the computations performed are based on the permuted inputs as well. Future work includes protocols supporting zero-knowledge proofs that are efficient at computation time (although perhaps expensive to verify). In addition, as mentioned previously, better dealing with premature termination of an AC-protocol would be also very interesting for future research.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago, Chile, September 12-15 1994. VLDB.

[2] Rakesh Agrawal and Evimaria Terzi. On honesty in sovereign information sharing. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT)*, 2006.

[3] Mikhail J. Atallah, Marina Bykova, Jiangtao Li, and Mercan Karahan. Private collaborative forecasting and benchmarking. In *Proc. 2d. ACM Workshop on Privacy in the Electronic Society (WPES)*, Washington, DC, October 28 2004.

[4] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In A.M. Odlyzko, editor, *Advances in Cryptography, CRYPTO86: Proceedings*, volume 263, pages 251–260. Springer-Verlag, Lecture Notes in Computer Science, 1986.

[5] 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):31–50, October 24 1995.

[6] Rosario Gennaro, Michael O. Rabin, and Tal 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*, pages 101–111, September 21 1998.

[7] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikainen. On secure scalar product computation for privacy-preserving data mining. In Choonsik Park and Seongtaek Chee, editors, *The 7th Annual International Conference in Information Security and Cryptology (ICISC 2004)*, Seoul, Korea, December 2-3 2004.

[8] O. Goldreich, S. Micali, and 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*, pages 218–229, 1987.

[9] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.

[10] Shafi Goldwasser, Silvio Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC'85)*, pages 291–304, Providence, Rhode Island, U.S.A., May 6-8 1985.

[11] Standard for privacy of individually identifiable health information. *Federal Register*, 67(157):53181–53273, August 14 2002.

[12] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[13] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 59–66, San Francisco, California. United States, 1998. ACM Press.

[14] Moni Naor, Benny Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*. ACM Press, 1999.

[15] Secure hash standard. Technical Report FIPS PUB 180-1, National Institutes of Standards and Technology, April 17 1995.

[16] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - Eurocrypt '98, LNCS 1403*, pages 308–318. Springer-Verlag, 1998.

[17] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[18] Ronald L. Rivest. The md5 message-digest algorithm. Technical Report RFC 1321, Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[19] Andrew C. Yao. Protocols for secure computation. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.

[20] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.

[21] Justin Zhan, Stan Matwin, and LiWu Chang. Privacy-preserving collaborative association rule mining. In *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, Storrs, Connecticut, August 7-10 2005.