

AC-Framework for Privacy-Preserving Collaboration*

Wei Jiang[†]

Chris Clifton[‡]

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 the field of privacy-preserving data mining. In this paper, we present the full spectrum of the accountable computing (AC) framework, which is sufficient or practical for many applications without the complexity and cost of an 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 [6], U.S. healthcare laws [13], 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 for mining without having to actually share or disclose data. This would maintain the security provided by separation of control while still obtaining the benefits of a global data source.

Secure multi-party computation (SMC) [9, 22, 23] 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 [22]: 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 [16], auctions [17] and among others.

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 [9]. 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. 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 an SSMC-protocol that searches for possible overlap of the two companies’ trucking routes. It

*The authors’ work is supported by National Science Foundation under Grant No. 0428168.

[†]Department of Computer Science, Purdue University, West Lafayette, IN 47907, wjiang@cs.purdue.edu.

[‡]Department of Computer Science, Purdue University, West Lafayette, IN 47907, clifton@cs.purdue.edu

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. (If the parties trusted each other, why not just share the data?) 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. 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.

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). In addition, there are situations where collaborating parties are required to follow certain regulations. Regulatory agencies may need to audit whether or not some regulations have been violated through spot checking to detect malicious behaviors. These situations lead us to the accountable computing (AC) framework.

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 could not have known, much less 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 (and punished through contractual or legal obligations). 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 to be run for auditing or 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 its practicality. Section 2 presents security definition and related ideas. Section 3 introduces the full spectrum of 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. 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 previously 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 [23]. This was extended to multiparty computations by Goldreich et al. [9]. They developed a framework for secure multiparty computation, and in [10] 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 2.1. *Let $f : \{0, 1\}^* \times \{0, 1\}^* \mapsto \{0, 1\}^* \times \{0, 1\}^*$ 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, \dots, m_t) (resp., (y, r, m_1, \dots, 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 $\text{OUTPUT}^\Pi(x, y) = (\text{OUTPUT}_1^\Pi(x, y), \text{OUTPUT}_2^\Pi(x, y))$.*

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

$$\begin{aligned} & \{(S_1(x, f_1(x, y)), f(x, y))\}_{x, y} \stackrel{C}{=} \\ & \{(\text{VIEW}_1^\Pi(x, y), \text{OUTPUT}_1^\Pi(x, y))\}_{x, y} \\ & \{(S_2(y, f_2(x, y)), f(x, y))\}_{x, y} \stackrel{C}{=} \end{aligned}$$

$$\{(\text{VIEW}_2^\Pi(x, y), \text{OUTPUT}^\Pi(x, y))\}_{x, y}$$

where $\stackrel{C}{\equiv}$ 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 [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.

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, 7] 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 [2] adopts a game-theoretic approach in that participants are rational, and when periodically using an auditing device are expected to provide truthful information.

A special case of the AC-framework is presented in [14] in that its verification process is prohibited from leaking any information regarding participating parties' private inputs. The paper also presents a general way to convert a semi-honest protocol to satisfy the specific framework. On the contrary, this paper presents the full spectrum of the AC-framework with detailed analyses, and we present a real life application to demonstrate its applicability and practicality.

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: 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 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. We next present essential definitions and key components that a protocol needs to guarantee under the AC-framework.

DEFINITION 3.1. (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** $\alpha \in \{\text{Participating-party}, \text{Third-party}\}$: *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 $\alpha = \text{Participating-party}$, $\Gamma_{\alpha \times \beta}$ 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 .

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 a 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.

Some secure function evaluation protocols utilize a trusted third party (TTP) to perform certain intermediate computations. The verifier in our framework is different from the TTP in those protocols since the intermediate computations performed by the TTP are required during the execution of the protocol and those computations directly affect the correctness of the protocol. Not only does this reduce the load on the third party, but to some extent less trust is required as the results are not dependent on that party.

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

DEFINITION 3.2. (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.*
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 refer to such a party as a verifier hereafter) 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 3.2 is optional for each run of an AC-protocol. Details follow later in the section.

3.1 General Assumptions. Here we clarify two key assumptions adopted throughout the framework: one is related to the nature of involved entities, and the other concerns the number of malicious parties allowed.

3.1.1 Nature of Involved Entities. In the AC-framework, the involved 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 this paper, we assume that there is one or a group of verifier(s) and that the verifier always behaves honestly during the verification process. In general, AC-protocols (including that in Section 4.3) prevent a malicious verifier from verifying a malicious party behaved correctly, or from proving an honest party behaved incorrectly. A malicious verifier could leave the state of verification undetermined, but as simple refusal to participate would accomplish this, this is impossible to prevent.

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, since we merely consider two-party protocols under the AC framework, we allow the situation where both participating parties are dishonest but without collusion between the two.¹ An AC-protocol should explicitly state a bound on the number of dishonest parties.

3.2 Guideline for Designing AC-Protocol. Based on Definition 3.2, 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 an SSMC-protocol, except that addi-

¹If two malicious parties can communicate freely, a third trusted party that is aware of all communications between the two may be needed to guarantee the integrity of their collaboration (i.e., under the anti-trust law).

- 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 3.2.

Figure 1: Guideline for Designing an AC-protocol

tional 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.

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 [8, 24], is to use a secure dot product protocol. To demonstrate the usefulness and applicability of the AC-framework, in this section, we present three protocols for finding frequent itemsets: SSMC-FFI, MSMC-FFI and AC-FFI. Based on empirical results, we analyze the pros and cons of each protocol. Before providing details, we first introduce the concept of homomorphic encryption used as building blocks for these protocols.

4.1 Homomorphic Encryption. 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., $\forall(r_1, x_1), (r_2, x_2) \in R \times X, \prod(E(r_1, x_1), E(r_2, x_2)) = E(r_3, x_1 + x_2)$, 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 [12]. 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 [19] has these properties. Honest behaviors in the malicious model can be enforced using efficient zero-knowledge (ZK) proofs [11] under the random oracle model [4]. In order to utilize ZK proof techniques, the Paillier's scheme has following additional properties (two-party case) [5]:

- 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 P1, then P1 can also efficiently generate ZK proof, denoted as $ZK(c)$, such that without disclosing M , P1 is able to prove to a verifier that c is the correct encryption of some value P1 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 combine 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 $E(r, M)$, $E(r', k \cdot M)$ can be efficiently computed along with its ZK proof.

Based on these properties, we present two protocols: SSMC-FFI and MSMC-FFI. We then show how to convert MSMC-FFI into AC-FFI, allowing the semi-honest behavior to be verified. For illustrative purpose, we use $E(M)$ and $D(c)$ to represent $E(r, M)$ and $D_{k_{pr}}(c)$ respectively wherever the context is clear in the rest of the paper.

4.2 SSMC-FFI and MSMC-FFI. The protocols presented in [8, 24] are basically a secure dot product protocol between two vectors whose entries are either 0 or 1 values. Here, we present a variant that can be used as subroutines to construct other privacy preserving protocols. 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 $\text{FFI}(\vec{v}_1, \vec{v}_2) \rightarrow (r_1, r_2)$, such that $r_1 + r_2 \pmod{N} = \vec{v}_1 \bullet \vec{v}_2$, where N is a large field and r_1, r_2 are uniformly distributed in $\{0, \dots, N-1\}$.

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

Algorithm 1 SSMC-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

{OUTPUT: P1 $\leftarrow r_1$ and P2 $\leftarrow r_2$ }

1: P1:

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

- (b). Send x_1, \dots, x_m to P2.

2: P2:

- (a). Compute $ESum \leftarrow \prod_{\forall i \wedge \vec{v}_2[i]=1} x_i$;
- (b). Compute $er_1 \leftarrow \prod(ESum, E(r_2))$, where r_2 is randomly chosen;
- (c). Send er_1 to P1.

3: P1:

- (a). Compute $r_1 \leftarrow D(er_1)$
-

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; the common characteristics among all these values is that their corresponding $\vec{v}_2[i]$ values must be 1. At the end, P1 computes a share of the actual dot

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

product value.

Table 1 shows a dataset vertically partitioned between P1 and P2, which 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 random shares of $\vec{v}_1 \bullet \vec{v}_2 = 3$ 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 [8, 24].

Key steps of the MSMC-FFI protocol are highlighted in Algorithm 2 (based on Protocol 2 presented in [15]). The main difference from the SSMC-FFI protocol is that ZK proofs are adopted to enforce honest behaviors. Steps 2(a) and 3(a) use ZK proofs to examine if received information is legitimate. In addition, step 3(c) is performed via threshold decryption and only P1 gets the decrypted value. Detailed security proof and correctness analysis can be found in [15].

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:

- Both parties perform *expected* computations;
- $\alpha =$ Third-party;
- β : Nothing is disclosed regarding each party's private input.

Note that $\Gamma_{\alpha \times \beta}$ implies that the verification phase of AC-FFI should be able to detect any malicious behavior (prevented under the malicious model) that occurs in the computation phase. Thus, the verification phase needs to verify every step in the computation phase.

Algorithm 2 MSMC-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, and each party has a private share of the decryption key;
{OUTPUT: P1 $\leftarrow r_1$ and P2 $\leftarrow r_2$ }

1: P1:

- (a). Encrypt \vec{v}_1 : $x_i \leftarrow E(\vec{v}_1[i])$, for $i = 1, \dots, m$, where r is randomly chosen for each $\vec{v}_1[i]$;
- (b). Compute $ZK(x_i)$ for each x_i ;
- (c). Send $I = \{\langle x_1, ZK(x_1) \rangle, \dots, \langle x_m, ZK(x_m) \rangle\}$ to P2.

2: P2:

- (a). **Abort**, if there exists an invalid ZK proof in I ;
- (b). Encrypt \vec{v}_2 : $y_i \leftarrow E(\vec{v}_2[i])$, for $i = 1, \dots, m$, where r is randomly chosen for each $\vec{v}_2[i]$;
- (c). Compute $ZK(y_i)$ for each y_i ;
- (d). Compute $z_i \leftarrow E((\vec{v}_1[i] \cdot \vec{v}_2[i]))$ (from x_i and y_i) and $ZK(z_i)$ for each z_i ;
- (e). Compute $er_2 \leftarrow E(r_2)$ and $ZK(er_2)$ of er_2 ;
- (f). Send $I^a = \{\langle y_1, ZK(y_1) \rangle, \dots, \langle y_m, ZK(y_m) \rangle\}$, $I^b = \{\langle z_1, ZK(z_1) \rangle, \dots, \langle z_m, ZK(z_m) \rangle\}$ and $I^c = \{\langle er_2, ZK(er_2) \rangle\}$ to P1.

3: P1:

- (a). **Abort**, if there exists an invalid ZK proof in either I^a , I^b or I^c ;
 - (b). Compute $ESum \leftarrow \prod_{V_i} z_i$;
 - (c). Compute $r_1 \leftarrow D(\prod(ESum, er_2))$ (Both parties jointly perform this decryption process, but only P1 gets to know r_1 value).
-

Designing such an AC-FFI protocol is straightforward given the MSMC-FFI protocol. Key steps of AC-FFI's computation phase are provided in Algorithm 3. The protocol adopts a secure signature scheme denoted by $Sign$ (e.g., RSA [20]) which is used during the verification process.² In order to fit the domain of a digital signature scheme, we also use a hash function (e.g., SHA

²For clarity, we assume an authenticated channel between P1 and P2 and show only signatures needed for verification; in practice and for any protocol presented in this paper, all messages would be signed to ensure integrity.

[18], MD5 [21]) to compute the hash value of an intended message and then sign the hash value instead. Algorithm 3 is identical to MSMC-FFI (Algorithm 2), except for the computation and verification of ZK proofs. Thus, we expect the computation phase of AC-FFI to be much more efficient than MSMC-FFI.

Algorithm 3 AC-FFI Protocol: Computation phase

Require: $\vec{v}_1, \vec{v}_2, E, Sign, H, ||$, where $|\vec{v}_1| = |\vec{v}_2| = m$ and E has certified public key parameters, and each party has a private share of the decryption key, $Sign$ is a secure signature scheme, H is a hash function and $||$ is a message concatenation operator
{OUTPUT: P1 $\leftarrow r_1$ and P2 $\leftarrow r_2$ }

1: P1:

- (a). Encrypt \vec{v}_1 : $x_i \leftarrow E(\vec{v}_1[i])$, for $i = 1, \dots, m$, where r is randomly chosen for each $\vec{v}_1[i]$;
- (b). Send $I[x_i] = \{x_1, \dots, x_m\}$ and $Sign_{P1}(H(I[x_i]))$ to P2.

2: P2:

- (a). **Abort**, if $Sign_{P1}(H(I[x_i]))$ is invalid;
- (b). Encrypt \vec{v}_2 : $y_i \leftarrow E(\vec{v}_2[i])$, for $i = 1, \dots, m$, where r is randomly chosen for each $\vec{v}_2[i]$;
- (c). Compute $z_i \leftarrow E((\vec{v}_1[i] \cdot \vec{v}_2[i]))$ from x_i and y_i ;
- (d). Compute $er_2 \leftarrow E(r_2)$;
- (e). Send $I^a[y_i] = \{y_1, \dots, y_m\}$, $I^b[z_i] = \{z_1, \dots, z_m\}$, $I^c[er_2] = \{er_2\}$ and $Sign_{P2}(H(I^a[y_i] || I^b[z_i] || I^c[er_2]))$ to P1.

3: P1:

- (a). **Abort**, if $Sign_{P2}(H(I^a[y_i] || I^b[z_i] || I^c[er_2]))$ is invalid;
 - (a). Compute $ESum \leftarrow \prod_{V_i} z_i$;
 - (b). Compute $r_1 \leftarrow D(\prod(ESum, er_2))$ (Both parties jointly perform this decryption process, but only P1 gets to know r_1 value).
-

The verification phase of AC-FFI is presented in algorithm 4. The verification phase is basically to examine the consistencies of all related ZK proofs as in the malicious model. Steps A and B in Algorithm 4 require P1 and P2 to send the messages they received during the computation phase, plus ZK proofs of messages they sent. At step C, the verifier first examines whether or not P1 and P2 send valid messages for veri-

Algorithm 4 AC-FFI protocol: Verification phase

Require: V, H, E , where V is a verifier, H and E are used in the computation phase

A P1: send $I[ZK(x_i)], I^a[y_i], I^b[z_i], I^c[er_2]$ and $Sign_{P2}(H(I^a[y_i]||I^b[z_i]||I^c[er_2]))$ to V ;

B P2: send $I^a[ZK(y_i)], I^b[ZK(z_i)], I^c[ZK(er_2)], I[x_i]$ and $Sign_{P1}(H(I[x_i]))$ to V ;

C V can detect malicious behaviors by performing the following steps sequentially:

1. P1 is malicious if $Sign_{P2}(H(I^a[y_i]||I^b[z_i]||I^c[er_2]))$ is invalid;
 2. P2 is malicious if $Sign_{P1}(H(I[x_i]))$ is invalid;
 3. P1 is malicious if $I[ZK(x_i)]$ and $I[x_i]$ are not consistent;
 4. P2 is malicious if either $\langle I^a[y_i], I^a[ZK(y_i)] \rangle, \langle I^b[z_i], I^b[ZK(z_i)] \rangle$ or $\langle I^c[er_2], I^c[ZK(er_2)] \rangle$ is not consistent.
-

fication. Then steps C(3) and C(4) check consistencies of all ZK proofs.

4.3.1 Security and Soundness of AC-FFI. Under the AC-framework, the computation phase of an AC-protocol is required to be secure only under the semi-honest model. In addition, because Algorithm 3 is directly derived from MSMC-FFI without computation and verification of ZK proofs, the computation phase of AC-FFI inherently satisfies the security requirement of the AC-framework. Next, we show that the verification phase of AC-FFI is sound.

CLAIM 4.1. *The verification phase of AC-FFI is sound (definition 3.2) provided that the computation phase of AC-FFI did not abort.*

Proof. Sketch: The first two steps of Algorithm 4 require both parties to send the messages received during the computation phase. Steps C(1) and C(2) check legitimacy of the messages. For instance, at step C(1), if the signature is not valid, the verifier can claim P1 did not behave honestly due to the assumption that the computation phase did not terminate prematurely. This implies that P1 should have received valid message and signature pairs. The same reasoning applies at step C(2).

Once no malicious behavior is detected at these two steps, the verifier can be certain that the messages received from P1 and P2 were actually used during the computation phase. Step C(3) examines if ZK proofs P1 provided are consistent with the messages (or

encryptions) computed during the computation phase. Any inconsistency can lead to a conclusion that P1 is malicious (or did not perform expected computations). Since the verification phase verifies the same set of ZK proofs as those in MSMC-FFI, it can detect any malicious behavior prevented under the malicious model. In addition, because these proofs are zero-knowledge, the verifier is unable to use them to infer information regarding parties' private inputs. Consequently, 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. Furthermore, the verifier can claim any party who refuses to participate in the verification process as a malicious entity. We next empirically analyze the complexity of the three protocols.

4.3.2 Empirical Analysis. In our experience with similar protocols, the time to perform homomorphic encryptions dominates the computational and communication times of such protocols. Therefore, we estimate the relative performance of the three FFI protocols based on the time to perform required number of encryptions and decryptions. The protocols are implemented in Java, and all executions were carried out on Solaris Sparc with two 1Ghz processors and 2GB memory. Because some values used in the encryption process are randomly chosen, there is a small variance among different executions. Thus, we report the average time complexity of each protocol across multiple executions. Figure 2 shows the empirical results.

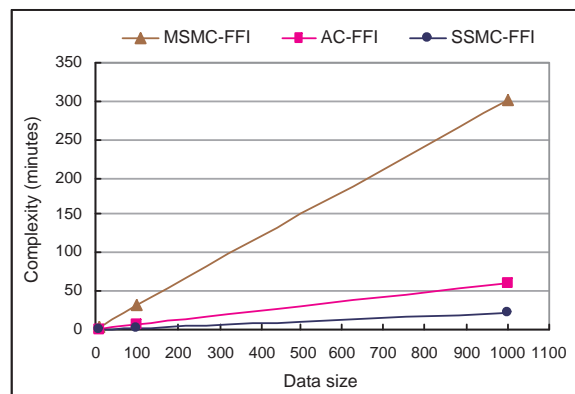


Figure 2: Time complexity

As shown in the figure, for each protocol, the time

complexity grows linearly with the size of the dataset. SSMC-FFI is about ten times faster than MSMC-FFI, and the computation phase of AC-FFI is about five times faster than MSMC-FFI since all ZK proofs are delayed to the verification phase. Although we did not show the complexity of the verification phase, its cost is near that of the MSMC-FFI protocol. These results confirm our intuition: the computation phase of an AC-protocol can be more efficient than an MSMC-protocol. Furthermore, since malicious behavior can be detected by verification, participating parties have an incentive to follow the protocol correctly; as a result, participating parties are more likely to behave correctly under the AC-framework (even if verification is only occasional or on demand) than with an SMC-protocol secure only 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. Equivalently, malicious behaviors could be detected under the framework.

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 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 is optional and 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-framework. Since the verification phase of AC-FFI does not leak any private information regarding participating parties' inputs, the verification process can be carried out repeatedly with different verifiers.

Therefore, the trust imposed on the verifier is limited. Future work includes protocols supporting more than two parties. As mentioned previously, dealing with premature termination of an AC-protocol would also be very interesting for future research.

Acknowledgment.

The authors wish to thank anonymous reviewers for their valuable suggestions, as well as Murat Kantarcioglu for his software implementation and insightful discussions.

References

- [1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*. Santiago, Chile: VLDB, Sept. 12-15 1994, pp. 487-499. <http://www.vldb.org/dblp/db/conf/vldb/vldb94-487.html>
- [2] R. Agrawal and E. Terzi, "On honesty in sovereign information sharing," in *Proceedings of the 10th International Conference on Extending Database Technology (EDBT)*, 2006.
- [3] M. J. Atallah, M. Bykova, J. Li, and M. Karahan, "Private collaborative forecasting and benchmarking," in *Proc. 2d. ACM Workshop on Privacy in the Electronic Society (WPES)*, Washington, DC, Oct. 28 2004.
- [4] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM Conference on Computer and Communications Security*, 1993, pp. 62-73. citeseer.ist.psu.edu/bellare95random.html
- [5] I. Damgard, M. Jurik, and J. Nielsen, "A generalization of paillier's public-key system with applications to electronic voting," 2003. citeseer.ist.psu.edu/damgard03generalization.html
- [6] "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*, vol. No I., no. 281, pp. 31-50, Oct. 24 1995. http://ec.europa.eu/justice_home/fsj/privacy/law/index_en.htm
- [7] R. Gennaro, M. O. Rabin, and T. Rabin, "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*, September 21 1998, pp. 101-111.
- [8] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen, "On secure scalar product computation for privacy-preserving data mining," in *The 7th Annual International Conference in Information Security and Cryptology (ICISC 2004)*, C. Park and S. Chee, Eds., Seoul, Korea, Dec. 2-3 2004.

- [9] 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*, 1987, pp. 218–229. <http://doi.acm.org/10.1145/28395.28420>
- [10] O. Goldreich, *The Foundations of Cryptography*. Cambridge University Press, 2004, vol. 2, ch. General Cryptographic Protocols. <http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/prot.ps>
- [11] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems," *Journal of ACM*, vol. 38, pp. 690–728, 1991.
- [12] S. Goldwasser, S. 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)*, Providence, Rhode Island, U.S.A., May 6-8 1985, pp. 291–304.
- [13] "Standard for privacy of individually identifiable health information," *Federal Register*, vol. 67, no. 157, pp. 53 181–53 273, Aug. 14 2002. <http://www.hhs.gov/ocr/hipaa/finalreg.html>
- [14] W. Jiang and Chris, "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, December 18-22 2006.
- [15] M. Kantarcioglu and O. Kardes, "Privacy-preserving data mining in malicious model," Department of Computer Science, Stevens Institute of Technology, Tech. Rep. CS-2006-06, 2006. <http://www.cs.stevens.edu/~onur/docs/TR-2006-06.pdf>
- [16] Y. Lindell and B. Pinkas, "Privacy preserving data mining," *Journal of Cryptology*, vol. 15, no. 3, pp. 177–206, 2002. http://www.research.ibm.com/people/l/lindell/id3_abs.html
- [17] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *Proceedings of the 1st ACM Conference on Electronic Commerce*. ACM Press, 1999.
- [18] "Secure hash standard," National Institutes of Standards and Technology, Tech. Rep. FIPS PUB 180-1, Apr. 17 1995. <http://www.itl.nist.gov/fipspubs/fip180-1.htm>
- [19] P. Paillier, "Public key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*. Springer-Verlag, 1999, pp. 223–238.
- [20] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," vol. 21, no. 2, pp. 120–126, 1978. <http://doi.acm.org/10.1145/359340.359342>
- [21] R. L. Rivest, "The md5 message-digest algorithm," Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc., Tech. Rep. RFC 1321, Apr. 1992.
- [22] A. C. Yao, "Protocols for secure computation," in *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*. IEEE, 1982, pp. 160–164.
- [23] —, "How to generate and exchange secrets," in *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*. IEEE, 1986, pp. 162–167.
- [24] J. Zhan, S. Matwin, and L. 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, Aug. 7-10 2005.