

---

## **Privacy-preserving data mining in the malicious model**

---

**Murat Kantarcioglu\***

Computer Science Department  
University of Texas at Dallas  
Dallas, TX, USA  
E-mail: muratk@utdallas.edu  
\*Corresponding Author

**Onur Kardes**

Computer Science Department  
Stevens Institute of Technology  
Hoboken, NJ, USA  
E-mail: onur@cs.stevens.edu

**Abstract:** Most of the cryptographic work in privacy-preserving distributed data mining deals with semi-honest adversaries, which are assumed to follow the prescribed protocol but try to infer private information using the messages they receive during the protocol. Although the semi-honest model is reasonable in some cases, it is unrealistic to assume that adversaries will always follow the protocols exactly. In particular, malicious adversaries could deviate arbitrarily from their prescribed protocols. Secure protocols that are developed against malicious adversaries require utilisation of complex techniques. Clearly, protocols that can withstand malicious adversaries provide more security. However, there is an obvious trade-off: protocols that are secure against malicious adversaries are generally more expensive than those secure against semi-honest adversaries only. In this paper, our goal is to make an analysis of trade-offs between performance and security in privacy-preserving distributed data mining algorithms in the two models. In order to make a realistic comparison, we enhance commonly used subprotocols that are secure in the semi-honest model with zero knowledge proofs to be secure in the malicious model. We compare the performance of these protocols in both models.

**Keywords:** privacy-preserving data mining; secure multiparty computation; malicious model.

**Reference** to this paper should be made as follows: Kantarcioglu, M. and Kardes, O. (xxxx) 'Privacy-preserving data mining in the malicious model', *Int. J. Information and Computer Security*, Vol. X, No. Y, pp.000–000.

**Biographical notes:** Murat Kantarcioglu is currently an Assistant Professor at the University of Texas at Dallas, USA. He obtained a PhD Degree from Purdue University in 2005. He received his Master's Degree in Computer Science from Purdue University in 2002 and his Bachelor's Degree in Computer Engineering from Middle East Technical University, Ankara, Turkey in 2000. His research interests lie at the intersection of privacy, security, data mining and databases.

Onur Kardes is a PhD student in the Computer Science Department at Stevens Institute of Technology in Hoboken, New Jersey, USA. He received an MS Degree in Computer Engineering from Bogazici University, Istanbul, Turkey in 2002 and a BS in Computer Science from Bilkent University, Ankara, Turkey in 2000. His research interests span information security issues, including cryptography, privacy and foundations of computer security.

---

## 1 Introduction

Data needed for many crucial data mining tasks is distributed among several parties with different security and privacy concerns. In many distributed data mining settings, disclosure of the original data sets is not acceptable due to privacy concerns. To address this problem, several privacy-preserving distributed data mining protocols using cryptographic techniques have been suggested. (See Section 2.1 for a detailed discussion of previous work.) Depending on the adversarial behaviour assumptions, those protocols use different models. In the semi-honest model, each party is assumed to follow the protocol without any deviation. Nevertheless, the assumption of semi-honest behaviour may be unrealistic in some settings. In such cases, participating parties may prefer to use a protocol that is secure against malicious behaviour. It is clear that the protocols secure in the malicious model offer more security, but this generally requires complex techniques to be achieved. As a result, those protocols are typically extremely inefficient as compared to the ones secure in the semi-honest model.

The efficiency versus security trade-off between the protocols that are secure in the malicious model and in the semi-honest model is not clear in practice. Therefore, a detailed analysis of the corresponding costs could be a valuable tool for decision-makers. In this paper, we investigate several protocols that are previously defined in the semi-honest model as primitives for different privacy-preserving data mining algorithms. We summarise these basic subprotocols and provide ways to make them secure against malicious adversaries using efficient zero knowledge proofs. Having basic subprotocols secure in the malicious model is neither sufficient nor necessary for constructing privacy-preserving data mining applications as this may require further study and utilisation of some other cryptographic tools. But, we believe that the methods we propose to make these primitives more efficient can also be applied to large-scale applications.

### 1.1 *Our contributions*

In this paper, we present two-party secure protocols in the malicious model for equality, dot product and full-domain set operations. For dot product, we define two different protocols that are secure against malicious adversaries: the first protocol is a straightforward extension of the semi-honest version using zero knowledge proofs; the second, more efficient protocol is specifically designed against malicious adversaries. We also prove that the given protocols are secure against malicious adversaries. Our initial results indicate that for efficiency purposes, it may be preferable to fully redesign

protocols in the malicious model instead of directly converting secure protocols in the semi-honest model to secure protocols in the malicious model using zero-knowledge proofs. We also provide extensive experimental analysis of the given protocols.

## 1.2 Organisation of the paper

In Section 2, we discuss the necessary background in privacy-preserving data mining and secure multi-party computation. In Section 3, we provide secure subprotocols in the malicious model along with their security analysis. In Section 4, we compare the relative efficiency of each protocol both in the semi-honest and the malicious model. Finally, we conclude the paper with the discussion of the current results and our planned future work in Section 5.

## 2 Background

In this section, we first discuss the previous work done in privacy-preserving data mining. Later, we describe the cryptographic tools and definitions used in this paper.

### 2.1 Related work

Many different distributed privacy-preserving data mining algorithms have been designed using cryptographic techniques. Usually one of two different assumptions about the distribution of the data is used in those protocols. In the case of horizontally partitioned data, different sites collect the same set of information about different entities. For example, different credit card companies may collect credit card transactions of different individuals. Secure distributed protocols have been developed for horizontally partitioned data for mining decision trees (Lindells and Pinkas, 2000), association rules (Kantarcioglu and Clifton, 2004a), k-means clustering (Lin *et al.*, 2005), k-nn classifiers (Kantarcioglu and Clifton, 2004b).

In the case of vertically partitioned data, we assume that different sites collect information about the same set of entities but they collect different feature sets. For example, both a university and a hospital may collect information about a student. Again, secure protocols for the vertically partitioned case have been developed for mining association rules (Vaidya and Clifton, 2002), decision trees (Du and Zhan, 2002) and k-means clusters (Jagannathan and Wright, 2005).

To the best of our knowledge, all of those previous protocols were proven or claimed to be secure only in the semi-honest model. Recently, Schuster *et al.* has addressed malicious behaviour in distributed association rule mining in data grids (Gilburd *et al.*, 2004). In Gilburd *et al.* (2004), instead of using standard cryptographic malicious models, the authors have developed a new security model for malicious adversaries and applied their model to association rule mining. Instead, in this paper, we follow the generic malicious model definitions from the cryptographic literature. We also focus on the security issues in the malicious model and provide the malicious versions of the subprotocols commonly used in previous privacy-preserving data mining algorithms.

## 2.2 Cryptographic background

Our definition of privacy-preserving data mining implies that nothing other than the final data mining result or what is implied by it, is revealed during the data mining process. This definition is equivalent to the ‘security’ definition used in Secure Multi-party Computation (SMC) literature. In this section, we give an overview of the cryptographic definitions and techniques that are used in this paper.

### 2.2.1 Homomorphic encryption

Many subprotocols used in privacy-preserving data mining algorithms such as secure  $\log(x)$  (Lindell and Pinkas, 2002), secure set operations (Kissner and Song, 2005), and secure dot product (Jagannathan and Wright, 2005) protocols use an additive homomorphic encryption. In order to provide a fair comparison, we also utilise secure additive homomorphic public key cryptosystem in our solutions against malicious adversaries.

Let  $E_{pk}(\cdot)$  denote the encryption function with public key  $pk$  and  $D_{pr}(\cdot)$  denote the decryption function with private key  $pr$ . A secure public key cryptosystem is called additive homomorphic if it satisfies the following requirements:

- given the encryption of  $m_1$  and  $m_2$ ,  $E_{pk}(m_1)$  and  $E_{pk}(m_2)$ , there exists an efficient algorithm to compute the public key encryption of  $m_1 + m_2$ , denoted  $E_{pk}(m_1 + m_2) := E_{pk}(m_1) +_h E_{pk}(m_2)$
- given a constant  $k$  and the encryption of  $m_1$ ,  $E_{pk}(m_1)$ , there exists an efficient algorithm to compute the public key encryption of  $km_1$ , denoted  $E_{pk}(km_1) := k \times_h E_{pk}(m_1)$ .

For the sake of completeness, we provide a brief definition of the additive homomorphic cryptosystem that we use in our experiments. Please refer to (Damgard *et al.*, 2003) for details. The Paillier (1999) cryptosystem, which is based on composite residuosity assumption, satisfies the above properties and can be defined as follows:

- *Key generation* – Let  $p$  and  $q$  be prime numbers where  $p < q$  and  $p$  does not divide  $q - 1$ . For the Paillier encryption scheme, we set the public key  $pk$  to  $n$  where  $n = p \cdot q$  and private key  $pr$  to  $(\lambda, n)$  where  $\lambda$  is the lowest common multiplier of  $p - 1, q - 1$ .
- *Encryption with the public key* – Given  $n$ , the message  $m$ , and a random number  $r$  from 1 to  $n - 1$ , encryption of the message  $m$  can be calculated as follows:  $E_{pk}(m) = (1 + n)^m \cdot r^n \pmod{n^2}$ . Also note that given any encrypted message, we can get a different encryption by multiplying it with some random  $r^n$ .
- *Decryption with the private key* – Given  $n$ , the cipher text  $c = E_{pk}(m)$ , we can calculate the  $D_{pr}(c)$  as follows:  $m = \frac{(c^\lambda \pmod{n^2}) - 1}{n} \lambda^{-1} \pmod{n}$  where  $\lambda^{-1}$  is the inverse of  $\lambda$  in modulo  $n$ .
- *Adding two ciphertexts* ( $+_h$ ) – Given the encryption of  $m_1$  and  $m_2$ ,  $E_{pk}(m_1)$  and  $E_{pk}(m_2)$ , we can calculate the  $E_{pk}(m_1 + m_2)$  as follows:

$$\begin{aligned}
 & E_{pk}(m_1).E_{pk}(m_2) \bmod n^2 \\
 &= ((1+n)^{m_1} r_1^n) \cdot ((1+n)^{m_2} r_2^n) \bmod n^2 \\
 &= ((1+n)^{m_1+m_2} \cdot (r_1 r_2)^n) \bmod n^2 \\
 &= E_{pk}(m_1 + m_2).
 \end{aligned}$$

We would like to emphasise that this addition will actually return  $E_{pk}(m_1 + m_2 \bmod n)$ .

- *Multiplying a ciphertext with a constant ( $k \times_h E_{pk}(m_1)$ )* – Given a constant  $k$  and the encryption of  $m_1, E_{pk}(m_1)$ , we can calculate  $k \times_h E_{pk}(m_1)$  as follows:

$$\begin{aligned}
 k \times_h E_{pk}(m_1) &:= E_{pk}(m_1)^k \bmod n^2 \\
 &= ((1+n)^{m_1} \cdot r_1^n)^k \bmod n^2 \\
 &= (1+n)^{km_1} \cdot r_1^{kn} \bmod n^2 \\
 &= E_{pk}(k.m_1).
 \end{aligned}$$

Also, we use efficient non-interactive zero-knowledge protocols in the random oracle model to prove that the actions taken by the parties are correct without revealing any other information (Cramer *et al.*, 2001). We briefly summarise those protocols below. The implementation details of those protocols for Paillier encryption can be found in Cramer *et al.* (2000):

- *Threshold decryption (two-party case)* – given the common public key  $pk$ , the private key  $pr$  corresponding to  $pk$  has been divided into two pieces  $pr_0$  and  $pr_1$ . There exists an efficient, secure protocol  $D_{pr_i}(E_{pk}(a))$  that outputs the random share of the decryption result  $s_i$  (similar to classic secret sharing schemes) along with the non-interactive zero knowledge proof  $POD(pr_i, E_{pk}(a), s_i)$  showing that  $pr_i$  is used correctly. Those shares can be combined to calculate the decryption result. Also any single share of the private key  $pr_i$  cannot be used to decrypt the ciphertext alone. In other words  $s_i$  does not reveal anything about the final decryption result. We also use a special version of a threshold decryption such that only one party learns the decryption result. Such a protocol could be easily implemented exploiting the fact that for any given  $E_{pk}(a)$ , the party that needs to learn the decryption result could generate  $E_{pk}(r1)$  and then both parties jointly decrypt the  $E_{pk}(a) +_h E_{pk}(r1)$ . Since only one party knows the  $r1$ , only that party can learn the correct decryption result.
- *Proving that you know a plaintext* – a party  $P_i$  can compute the zero knowledge proof  $POK(e_a)$  if he knows an element  $a$  in the domain of valid plaintexts such that  $D_{pr}(e_a) = a$ .
- *Proving that multiplication is correct* – assume that party  $P_i$  is given an encryption  $E_{pk}(a)$  and chooses constant  $c$  and calculates  $E_{pk}(a.c)$ . Later on,  $P_i$  can give zero knowledge proof  $POMC(e_a, e_c, e_{a.c})$  such that  $D_{pr}(e_{a.c}) = D_{pr}(e_c).D_{pr}(e_a)$ .

In our security proofs, we use the simulators for the above zero-knowledge proofs guaranteed due to their security properties. As discussed in Cramer *et al.* (2001), those simulators return a state of the adversary that is statistically indistinguishable from the

state of the adversary in the real-life execution. Also, those simulators return the secret inputs used by the adversary for valid zero-knowledge proofs with overwhelming probability. We use both of these properties in our security proofs.

### 3 Secure protocols in the malicious model

All the protocols mentioned in this section are implemented in the semi-honest model as primitives for different privacy-preserving data mining algorithms. Here we summarise these basic subprotocols and provide ways to make them secure in the malicious model by using efficient zero-knowledge proofs discussed above.

#### 3.1 Secure equality protocol

One of the most common tools needed in many privacy-preserving data mining algorithms is to calculate whether two items are equal or not without revealing these items. Such secure equality protocol could easily be implemented using homomorphic encryption in the semi-honest model without using threshold decryption. The idea is simple. Assume that we have two parties  $P_0$  and  $P_1$ .  $P_0$  generates a homomorphic key pair and sends  $pk$  to  $P_1$  along with the item  $E_{pk}(x_0)$ . Given  $pk$ ,  $P_1$  calculates  $E_{pk}(-1.(x_1))$  where  $x_1$  is the  $P_1$ 's item. After that  $P_1$  calculates  $e = (E_{pk}(-1.(x_1)) +_h E_{pk}(x_0)) \times_h r = E_{pk}((x_0 - x_1).r)$  where  $r$  is a random number and sends  $e$  to  $P_0$ . Clearly if  $x_0 = x_1$  then  $D_{pr}(e) = 0$  else  $D_{pr}(e)$  is a random number. Although the above protocol is correct in the semi-honest model, it does not work in the malicious model because  $P_1$  can choose  $r = 0$  to make the decryption value 0.

We can develop an equality protocol that is secure in the malicious model using the threshold version of the homomorphic encryption for two-party case. The basic idea in the malicious version is that the two parties jointly calculate  $E_{pk}((x_0 - x_1).(r_0 + r_1))$  where  $r_0, r_1$  are random numbers chosen by the respective parties. Here at each step, using the zero knowledge protocols described above,  $P_0$  and  $P_1$  prove that the actions they have taken are consistent with the protocol without revealing anything.

In this protocol, each party sends the encrypted  $x_i$  value to the other party. Since each party only has the share of the private key, they cannot decrypt the other party's encrypted  $x_i$  value. Later on, each party calculates  $E_{pk}((x_0 - x_1).r_i)$  and proves to the other party that the calculation is correct using the zero knowledge proof of correct multiplication described in Section 2.2.1. After that each party calculates  $E_{pk}((x_0 - x_1).(r_0 + r_1))$  and jointly decrypts  $E_{pk}((x_0 - x_1).(r_0 + r_1))$  to learn  $(x_0 - x_1)(r_0 + r_1)$ . Clearly if the decrypted value is zero with very high probability  $x_0 = x_1$  else it means that  $x_0 \neq x_1$ . The details of the secure equality protocol in the malicious model is given in Protocol 3.1. Please note that we use an oracle call to do threshold decryption, instead of an actual implementation. As stated before, such oracle calls could be replaced with actual secure implementation. In Theorem A.5, we prove that the Protocol 3.1 is secure in the malicious model.

#### 3.2 Secure dot product

Secure dot product is another useful subprotocol that is commonly used in many privacy-preserving data mining algorithms. Using homomorphic encryption, it is straightforward to develop a secure dot product protocol in the semi-honest model.

Similarly,  $P_0$  generates a homomorphic key pair and sends the  $pk$  to  $P_1$  along with the encrypted vector  $(E_{pk}(\vec{x}_0) = (E_{pk}(x_{00}), E_{pk}(x_{01}), \dots, E_{pk}(x_{0n})))$ . Given  $pk$ ,  $P_1$  calculates  $e_{\vec{x}_0, \vec{x}_1} = (E_{pk}(x_{00}) \times_h x_{10}) +_h (E_{pk}(x_{01}) \times_h x_{11}) +_h \dots +_h (E_{pk}(x_{0n}) \times_h x_{1n})$  where  $x_{1i}$  is the  $P_1$ 's input vector's  $i$ -th component and sends  $e_{\vec{x}_0, \vec{x}_1} +_h E_{pk}(r_1)$  to  $P_0$ . By decrypting the  $P_1$ 's message,  $P_0$  learns the random share of the dot product result  $\vec{x}_0 \cdot \vec{x}_1 + r_1$ . Clearly  $P_0$  and  $P_1$  can combine their shares to learn  $\vec{x}_0 \cdot \vec{x}_1$ .

<p><b>Protocol 3.1</b>    Secure equality in the malicious model using threshold decryption</p> <p><b>Require:</b> Two parties <math>P_0</math> and <math>P_1</math> with the shares <math>pr_0</math> and <math>pr_1</math> of the private key and private inputs <math>x_0</math> and <math>x_1</math>.</p> <p><b>Ensure:</b> Return 1 if <math>x_0 = x_1</math> else return 0</p> <p><b>for all</b> <math>P_i</math> <b>do</b></p> <p style="padding-left: 2em;">Calculate <math>e_{x_i} = E_{pk}(x_i)</math></p> <p style="padding-left: 2em;">Create <math>POK(e_{x_i})</math></p> <p style="padding-left: 2em;">Send <math>(e_{x_i}, POK(e_{x_i}))</math> to <math>P_{1-i}</math></p> <p><b>end for</b></p> <p><b>for all</b> <math>P_i</math> <b>do</b></p> <p style="padding-left: 2em;">Check <math>POK(e_{x_{1-i}})</math> is valid else <b>ABORT</b></p> <p style="padding-left: 2em;">Calculate <math>e_{x_0-x_1} = e_{x_0} +_h (-1 \times_h e_{x_1})</math></p> <p style="padding-left: 2em;">Choose non-zero random <math>r_i</math> and calculate <math>e_{r_i} = E_{pk}(r_i), e_{(x_0-x_1), r_i} = e_{x_0-x_1} \times_h r_i</math></p> <p style="padding-left: 2em;">Create <math>POMC(e_{x_0-x_1}, e_{r_i}, e_{(x_0-x_1), r_i})</math></p> <p style="padding-left: 2em;">Send <math>(e_{x_0-x_1}, e_{r_i}, e_{(x_0-x_1), r_i}), POK(e_{r_i}), POMC(e_{x_0-x_1}, e_{r_i}, e_{(x_0-x_1), r_i})</math> to <math>P_{1-i}</math></p> <p><b>end for</b></p> <p><b>for all</b> <math>P_i</math> <b>do</b></p> <p style="padding-left: 2em;">Check whether the <math>e_{x_0-x_1}</math> sent by <math>P_{1-i}</math> is correct else <b>ABORT</b></p> <p style="padding-left: 2em;">Check whether <math>POK(e_{r_{1-i}})</math> is valid else <b>ABORT</b></p> <p style="padding-left: 2em;">Check whether <math>POMC(e_{x_0-x_1}, e_{r_{1-i}}, e_{(x_0-x_1), r_{1-i}})</math> valid else <b>ABORT</b></p> <p style="padding-left: 2em;">Calculate <math>e_{(x_0-x_1), (r_0+r_1)} = e_{(x_0-x_1), r_1} +_h e_{(x_0-x_1), r_0}</math></p> <p><b>end for</b></p> <p><b>for all</b> <math>P_i</math> <b>do</b></p> <p style="padding-left: 2em;">Jointly use the trusted party <math>T</math> to get <math>D_{pr}(e_{(x_0-x_1), (r_0+r_1)})</math></p> <p style="padding-left: 2em;">If <math>(x_0 - x_1) \cdot (r_0 + r_1) = 0</math> then return 1 else return 0</p> <p><b>end for</b></p>
--

Now we show how to securely evaluate the dot product in the malicious model for two-party case using threshold homomorphic encryption. We provide two different secure dot product protocols that can be used in the malicious model. The first protocol given in Section 3.2.1 is a generic extension of the described protocol in the semi-honest model using appropriate zero-knowledge proofs. Later on, in Section 3.2.2, we provide a solution that is more efficient than the one described in Section 3.2.1.

Our second protocol indicates that using generic transformation techniques to convert protocols from semi-honest to malicious model may not be efficient.

### 3.2.1 *Converting secure dot protocol in the semi-honest model to malicious model*

If we look at the dot product protocol in the semi-honest model carefully, we need to make sure that the  $P_1$  does the multiplications correctly (using zero knowledge proofs of plaintext knowledge) and all the encryptions sent are valid (using zero knowledge proofs of correct multiplication). These could be easily achieved using the zero knowledge protocols described in Section 2.2.1.  $P_0$  sends the encrypted values along with the associated proofs of correct encryption to  $P_1$ . For each multiplication,  $P_1$  generates the zero knowledge proof of correct multiplication and sends those to  $P_0$ .  $P_0$  can check those proofs to make sure that dot product calculated correctly. The details are described in Protocol 3.2. Also in Theorem A.6, we prove that the protocol is secure in the malicious model.

### 3.2.2 *Secure and efficient dot product for malicious model*

In the previous secure dot protocol,  $P_1$  is required to prove that each multiplication is computed correctly. Actually, all we need to do is to check whether the final result is correct (*i.e.*, whether  $r_0$  is calculated correctly). This is possible since at least one of the parties will behave semi-honestly. If both parties are malicious, we do not care whether the privacy of any party is protected or parties get correct results. Assuming that at least one party will behave in a semi-honest fashion (the other party can do any malicious act), we can give a more efficient protocol. We stress that our assumption of at least one party is semi-honest is consistent with the definitions of the malicious model. Therefore, our assumption does not reduce the security guarantees provided by the malicious model.

Assuming that at least one party is semi-honest, we know that  $r_0 = \sum_{i=0}^n (x_{0i} \cdot x_{1i}) + r_1$  is evaluated correctly by at least one party. Please note that both  $P_0$  and  $P_1$  have enough information to calculate  $r_0$ . If both  $P_0$  and  $P_1$  calculate the same  $r_0$  value then calculations must be correct, because at least one of them is semi-honest and calculates correct  $r_0$ . Therefore, if we securely make sure that both parties calculate the same value, then either of the local calculations could be decrypted to reveal  $r_0$  to  $P_0$ . In our second protocol, each party sends the encrypted inputs along with the knowledge of plaintext proofs to each other, then each party  $P_i$  locally computes its respective  $e_{r_0^i} = E_{pk}(r_0^i)$ . After that point, they use a slightly modified version of the equality protocol to check whether  $D_{pr}(e_{r_0^0}) = D_{pr}(e_{r_0^1})$  or not. If those two values are equal, both parties jointly decrypt one of those values to reveal  $r_0$  to  $P_0$ . Clearly, in this version, we do not need to send expensive zero-knowledge proofs of correct multiplications for every multiplication. Due to reduced number of zero-knowledge proofs, the following protocol can offer huge

savings when the vectors used for the dot product have many components. We provide the details of the efficient secure dot product function in Protocol 3.3. In Theorem A.7, we prove that the protocol is secure in the malicious model.

<p><b>Protocol 3.2</b> Secure dot product in the malicious model using threshold decryption: extension of the semi-honest version</p>
<p><b>Require:</b> Two parties <math>P_0</math> and <math>P_1</math> with the shares <math>pr_0</math> and <math>pr_1</math> of the private key <math>pr</math> and <math>n</math> bit vectors <math>\vec{x}_i</math> where <math>\vec{x}_i</math> belongs to <math>P_i</math>.</p> <p><b>Ensure:</b> Return <math>r_0 = \sum_i^n (x_{0i} \cdot x_{1i}) + r_1</math> to <math>P_0</math> and <math>r_1</math> to <math>P_1</math></p> <p><b>for</b> <math>P_0</math> <b>do</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, set <math>e_{x_i} = E_{pk}(x_{0i})</math> and create <math>POK(e_{x_i})</math></p> <p style="padding-left: 2em;">Send encryptions and non-interactive zero knowledge proofs to <math>P_1</math></p> <p><b>end for</b></p> <p><b>for</b> <math>P_1</math> <b>do</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, check whether <math>POK(e_{x_{0i}})</math> is correct else <b>ABORT</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, calculate <math>e_{x_{1i}} = E_{pk}(x_{1i})</math>, <math>e_{x_{0i} \cdot x_{1i}} = e_{x_{0i}} \times_h x_{1i}</math></p> <p style="padding-left: 2em;">Choose non-zero random <math>r_1</math></p> <p style="padding-left: 2em;">Calculate <math>e_{r_1} = E_{pk}(r_1)</math> and <math>POK(e_{r_1})</math></p> <p style="padding-left: 2em;">Calculate <math>e_s = E_{pk}\left(\sum_i^n (x_{0i} \cdot x_{1i}) + r_1\right) = e_{x_{00} \cdot x_{10}} +_h e_{x_{01} \cdot x_{11}} +_h \dots +_h e_{x_{0n} \cdot x_{1n}} +_h e_{r_1}</math></p> <p style="padding-left: 2em;"><math>\forall i</math>, send <math>e_{x_i}</math>, <math>e_{x_{0i} \cdot x_{1i}}</math>, <math>POK(e_{x_{0i}})</math>, <math>POMC(e_{x_{0i}}, e_{x_{1i}}, e_{x_{0i} \cdot x_{1i}})</math>, <math>e_{r_1}</math>, <math>POK(e_{r_1})</math> and <math>e_s</math> to <math>P_0</math></p> <p><b>end for</b></p> <p><b>for</b> <math>P_0</math> <b>do</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, check whether the <math>POK(e_{x_{0i}})</math> is correct else <b>ABORT</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, check whether the <math>POMC(e_{x_{0i}}, e_{x_{1i}}, e_{x_{0i} \cdot x_{1i}})</math> is correct else <b>ABORT</b></p> <p style="padding-left: 2em;">Calculate <math>e_s = E_{pk}\left(\sum_i^n (x_{0i} \cdot x_{1i}) + r_1\right)</math></p> <p><b>end for</b></p> <p>Jointly, call private decrypt function such that only <math>P_0</math> learns the decryption of <math>e_s</math></p>

### 3.3 Secure set operations

Secure set operations using homomorphic encryption have been proposed in the literature earlier in Freedman *et al.* (2004). Recently Dawn *et al.* showed how to use homomorphic encryption and the zero knowledge proofs for constructing secure set operation protocols in the malicious model (Kissner and Song, 2005). Given that  $P_0$  has  $n$  items and  $P_1$  has  $m$  items, the protocols described in Kissner and Song (2005) require  $O(nm)$  homomorphic encryptions and zero knowledge proofs of correct multiplication. Those protocols are quite efficient if the total item domain size ( $D$ ) is much bigger than the number of items

possessed by  $P_0$  and  $P_1$  (i.e.,  $D > n.m$ ). For the cases where both  $m$  and  $n$  are bigger than  $O(\sqrt{D})$  or where  $n$  or  $m$  equal to  $D$ , we suggest using the simple secure set intersection and set union protocols that are secure in the malicious model. Our algorithms require  $O(D)$  homomorphic encryptions and zero knowledge proofs of correct multiplication. The main idea is that we can represent the sets owned by each party as a bit vector of size  $D$  and use secure multiplication property of the homomorphic encryption and associated zero knowledge proof to give secure set protocols in the malicious model.

**Protocol 3.3** Secure dot product in the malicious model using threshold decryption:  
efficient dot product specific version

**Require:** Two parties  $P_0$  and  $P_1$  with the shares  $pr_0$  and  $pr_1$  of the private key  $pr$  and  $n$  bit vectors  $\vec{x}_i$  where  $\vec{x}_i$  belongs to  $P_i$ .

**Ensure:** Return  $r_0 = \sum_i^n (x_{0i} \cdot x_{1i}) + r_1$  to  $P_0$  and  $r_1$  to  $P_1$

**for all**  $P_i$  **do**

$\forall j$ , set  $e_{x_{ij}} = E_{pk}(x_{ij})$  and create  $POK(e_{x_{ij}})$

**if**  $P_i = P_1$  **then**

Choose random  $r_1$ , set  $e_{r_1} = E_{pk}(r_1)$  and create  $POK(e_{r_1})$

**end if**

Send encryptions and non-interactive zero knowledge proofs to  $P_{1-i}$

**end for**

**for**  $P_0$  **do**

$\forall i$ , check whether the  $POK(e_{x_{1i}})$  is correct else **ABORT**

Check whether the  $POK(e_{r_1})$  is correct else **ABORT**

Set  $e_{r_0}$  to  $E_{pk}\left(\sum_i^n (x_{0i} \cdot x_{1i}) + r_1\right) = e_{x_{00} \cdot x_{10}} +_h e_{x_{01} \cdot x_{11}} +_h \dots +_h e_{x_{0n} \cdot x_{1n}} +_h e_{r_1}$

**end for**

**for**  $P_1$  **do**

$\forall i$ , check whether the  $POK(e_{x_{0i}})$  is correct else **ABORT**

Set  $e_{r_1}$  to  $E_{pk}\left(\sum_i^n (x_{0i} \cdot x_{1i}) + r_1\right) = e_{x_{00} \cdot x_{10}} +_h e_{x_{01} \cdot x_{11}} +_h \dots +_h e_{x_{0n} \cdot x_{1n}} +_h e_{r_1}$

**end for**

Jointly call decrypt equality protocol to check whether  $D_{pr}(e_{r_1}) = D_{pr}(e_{r_0})$

**for all**  $P_i$  **do**

**if** Secure equality protocol returns true for  $D_{pr}(e_{r_1}) = D_{pr}(e_{r_0})$  **then**

Jointly call private decrypt protocol such that  $P_0$  learns the  $D_{pr}(e_{r_1})$

**end if**

**end for**

<p><b>Protocol 3.4</b> Secure set intersection in the malicious model using threshold decryption</p> <p><b>Require:</b> Two parties <math>P_0</math> and <math>P_1</math> with the shares <math>pr_0</math> and <math>pr_1</math> of the private key <math>pr</math> and input bit vectors of size <math>D</math> where <math>x_{ij}</math> is set to one if <math>P_i</math> has item <math>j</math></p> <p><b>Ensure:</b> Return <math>D</math> bit vector <math>I</math> that represents the set intersection where <math>I_i</math> is set to one if item <math>i</math> is in the set intersection.</p> <p><b>for</b> <math>P_0</math> <b>do</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, set <math>e_{x_{0i}} = E_{pk}(x_{0i})</math> and create <math>POK(e_{x_{0i}})</math> to prove that each <math>x_{0i}</math> is either zero or one</p> <p style="padding-left: 2em;">Send encryptions and non-interactive zero knowledge proofs to <math>P_1</math></p> <p><b>end for</b></p> <p><b>for</b> <math>P_1</math> <b>do</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, check whether <math>POK(e_{x_{0i}})</math> is correct else <b>ABORT</b></p> <p style="padding-left: 2em;"><math>\forall i, x_{1i}</math> calculate <math>e_{x_{1i}} = E_{pk}(x_{1i})</math>, <math>e_{x_{0i} \cdot x_{1i}} = e_{x_{0i}} \times_h x_{1i}</math></p> <p style="padding-left: 2em;"><math>\forall i</math>, send <math>e_{x_{0i}}, e_{x_{0i} \cdot x_{1i}}, POK(e_{x_{0i}})</math> (again proving <math>x_{1i}</math> is either zero or one), and <math>POMC(e_{x_{0i}}, e_{x_{1i}}, e_{x_{0i} \cdot x_{1i}})</math> to <math>P_0</math></p> <p><b>end for</b></p> <p><b>for</b> <math>P_0</math> <b>do</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, check whether the <math>POK(e_{x_{0i}})</math> is correct else <b>ABORT</b></p> <p style="padding-left: 2em;"><math>\forall i</math>, check whether the <math>POMC(e_{x_{0i}}, e_{x_{1i}}, e_{x_{0i} \cdot x_{1i}})</math> is correct else <b>ABORT</b></p> <p><b>end for</b></p> <p><math>\forall i</math>, jointly call threshold decryption function to learn <math>D_{pr}(e_{x_{0i} \cdot x_{1i}})</math>.</p> <p>set <math>I_i</math> to <math>D_{pr}(e_{x_{0i} \cdot x_{1i}})</math>.</p>
--

Let us assume that  $x_{0i}$  is set to 1 if  $P_0$  has item  $i$  in its private set else it is set to 0 (similarly for  $x_{1i}$  for  $P_1$ ). Clearly for calculating set intersection, we need to calculate  $x_{0i} \wedge x_{1i}$  for each  $i$ . Similarly, for set union, we need to calculate  $x_{0i} \vee x_{1i}$  for all  $i$ . Note that  $\wedge$  operation is just a multiplication. For the set union, we can rewrite  $x_{0i} \vee x_{1i}$  as  $\neg(\neg x_{0i} \wedge \neg x_{1i})$ . This implies that if  $(\neg x_{0i} \wedge \neg x_{1i})$  is equal to zero then item  $i$  is in the set union. Therefore, we can use the multiplication protocol for set union too. The details of the set intersection protocol which is very similar to dot product Protocol 3.2 is given in Protocol 3.4. The same protocol can be used for two-party set union using  $\neg x_{0i}$  and  $\neg x_{1i}$  as the input values and negating the output bits. As before, we can prove that the Protocol 3.4 is secure in the malicious model. Please see Theorem A.8 for details.

### 3.4 Secure comparison

Secure comparison is another important subprotocol that is commonly used in many different privacy-preserving data mining protocols. Since the most efficient protocols follow the generic circuit evaluation methods, generic conversion techniques could be used for creating secure comparison in the malicious model. Please see Malkhi *et al.* (2004) for details.

## 4 Performance evaluation

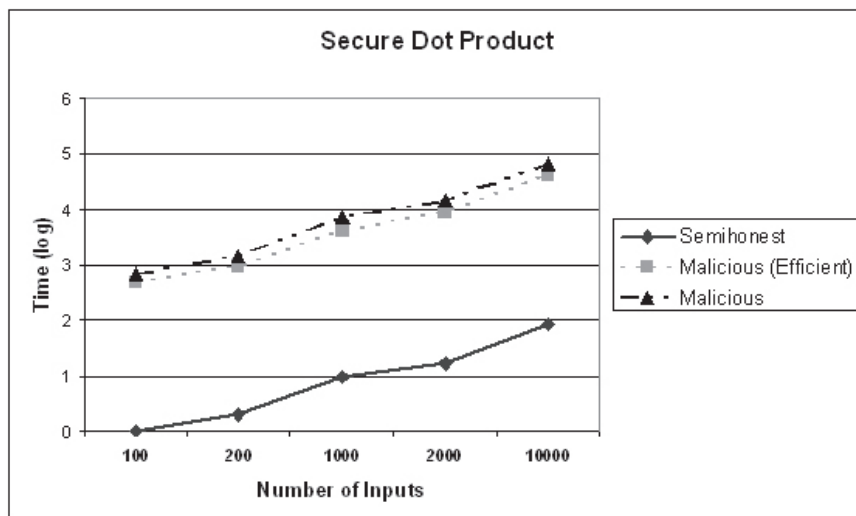
In this section, we analyse the performance of several privacy-preserving data mining algorithms in the malicious model. As stated before, the efficiency of a distributed data mining algorithm can be estimated in terms of the primitives it utilises; *i.e.*, the number of secure dot products, secure comparisons, *etc.* Therefore, in this section, we explore the efficiency trade-offs in these basic subprotocols and use the performance results to estimate possible overall slow down in the privacy-preserving distributed data mining algorithms in the malicious model. In our implementation, we used the zero knowledge protocols given for Paillier encryption in random oracle model (Cramer *et al.*, 2000).

### 4.1 Secure dot product

The running time of a secure dot product protocol can be expressed as  $O(n)$  where  $n$  is the size of the input dataset. Therefore, the efficiency of the protocol is highly dependent on the size of the input dataset. Also the running time of the protocol depends on the encryption key length (Kardes *et al.*, 2005).

To evaluate the performance of the secure dot product protocol in the malicious model and to show how the efficiency is effected by input size, we implement the protocols that are described Section 3.2.1, and Section 3.2.2. As a benchmark for efficiency loss, we also provide an implementation of the secure dot product protocol in the semi-honest model. The comparison results are presented in Figure 1.

**Figure 1** Performance evaluation results for the secure dot product protocol



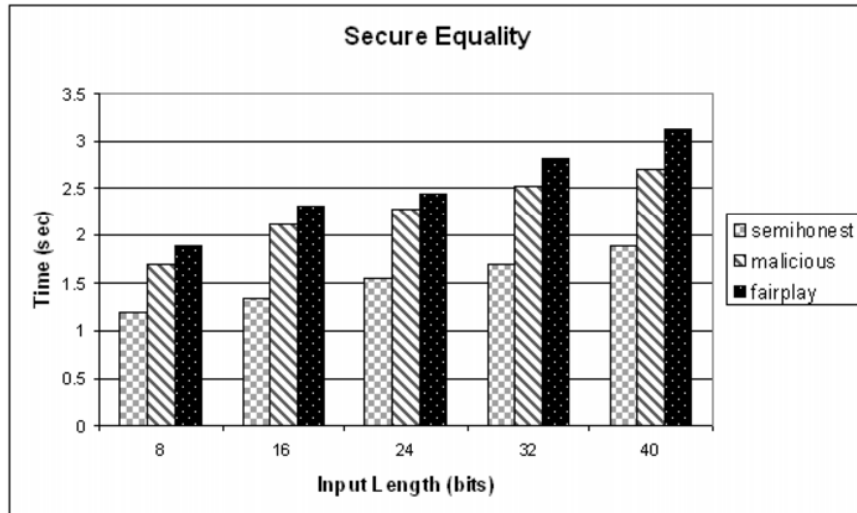
In Figure 1 the values in y-axis are displayed in  $\log_{10}$ . We show that there is a positive linear relationship between the overall running time of the secure dot product and the input size. Also it can be observed from Figure 1 that there is a significant difference (about 700 times) between running times of the secure dot product protocol in the semi-honest and malicious models. The running times displayed in the figure involve both communication and computation times and the difference can be explained with the impact of communication and computation overhead that the zero knowledge proofs bring. However such considerable difference should not lead to a conclusion that utilising the secure dot product protocol in the malicious model is totally unusable and inapplicable; in Figure 1 we also show that the overall running time in the malicious model can be cut by half by designing a more efficient protocol. It is possible that more sophisticated methods and modifications may lead to even bigger increases in efficiency.

Generic circuit evaluation methods (Malkhi *et al.*, 2004) can be an alternative for using zero knowledge proofs as they also provide security and privacy in the malicious model. However the size of the circuit to be used for evaluating the secure dot product protocol is proportional to the input size; *i.e.*, the circuit should be duplicated  $n$  times. As a result of this, for even small data sets (1000 elements), the circuit evaluation becomes totally infeasible to use.

#### 4.2 Secure equality check

Secure equality is an important and commonly used primitive in privacy-preserving data mining applications. In the basic secure equality protocol, which is also known as the Socialist Millionaires problem, two values are compared to output = or  $\neq$  without revealing the values (Boudot *et al.*, 2001).

**Figure 2** Performance evaluation results for the secure equality check protocol



For both primitives, since the number of inputs is fixed (as 2), the bit lengths of the inputs play a more important role on the overall running times. Such positive linear relationship can be observed in Figure 2. In that figure, we compare semi-honest and malicious

models; and we also provide running times for circuit evaluation using Fairplay (Malkhi *et al.*, 2004). It can be observed that the overall running time of our protocol in the malicious model is very close to one in the semi-honest model and significantly better than the circuit evaluation. This is because the number of zero knowledge proofs used in the malicious model does not depend on the bit length; however the size of the circuit used in Fairplay grows as the bit length increases.

#### 4.3 *Set union and intersection*

As it was explained in Section 3.3 the set intersection protocol can easily be converted into a secure dot product protocol where the number of inputs depend on the domain size. In addition to this, the inputs are bit vectors; which allow parties to generate encryptions for 0s and 1s in advance. Other than this property, our set intersection protocol is very similar to the dot protocol described in Section 3.2.1. Therefore, we expect those two protocols to have similar running times. On the other hand, secure set union protocol can be expressed as a set intersection protocol with negations.

#### 4.4 *Privacy-preserving k-means clustering algorithm using arbitrarily partitioned data*

K-means clustering is a simple and very commonly used clustering algorithm in data mining. It starts with an unclustered dataset with  $n$  elements and one attributes and outputs cluster assignments of each data element in the set. It requires prior knowledge of the number of clusters ( $k$ ). There are some number of recent studies (Paillier, 1999; Jagannathan and Wright, 2005) on privacy-preserving k-means clustering algorithm in the semi-honest model, where the input dataset is partitioned among two parties. The protocol described in Jagannathan and Wright (2005) uses arbitrarily partitioned data where any data element and/or attribute may be partitioned among parties. The attribute names and size of the data set are known to both parties.

The k-means algorithm involves mainly two subprotocols (Jagannathan and Wright, 2005): secure dot product and secure equality. The privacy-preserving algorithm utilises secure dot product protocol while calculating cluster centres and making cluster assignments. For each dot product, input sets of size 1 are used where 1 is the number of attributes, which is usually less than 20. Thus we would expect running times much lower than the ones displayed in Figure 1. In addition to this, the secure equality protocol is run only once per iteration of the k-means algorithm. We can argue that the secure dot product and equality protocols that we defined in Section 3 can be used as subprotocols in a privacy-preserving k-means algorithm in the malicious model efficiently. Conversion from semi-honest model to malicious model may bring additional complexities for the k-means algorithm, which will not be discussed here. However in an ordinary setting with ten attributes and five clusters k-means clustering in the malicious model takes more than 300 times longer than it takes in the semi-honest model.

#### 4.5 *Other PPDM applications in vertically partitioned data*

Some recent studies (Vaidya and Clifton, 2002; Yang and Wright, 2006) provide privacy-preserving association rule mining algorithms using vertically partitioned data (*i.e.*, the attributes (columns) of the data set is distributed among parties). For example,

(Yang and Wright, 2006) provides privacy-preserving protocol for Bayesian Network construction in the semi-honest model. The protocol defines efficient computation of association rules for Bayesian Networks. The computations involve secure dot products with inputs of length  $n$  where  $n$  can be arbitrarily large. The protocol also contains several other subprotocols which do not depend on  $n$  (the input size). When a conversion from semi-honest model to malicious model is thought, according to Figure 1 even our efficient dot product protocol for the malicious model does not produce practical results for  $n > 1000$ .

## 5 Conclusions

Most of the privacy-preserving data mining algorithms in the literature were developed against semi-honest adversaries; and their correctness and security rely on the presumption that all participating parties follow the protocol without any deviation. Although this assumption may be justifiable for some cases, there are many real-life examples where less restrictive assumptions should be made. The malicious model introduces solid rules for security while requiring significant computational and communicational overhead. However, the efficiency versus security trade off between semi-honest and malicious models is not clear in practice.

In this study, using standard primitive algorithms in the semi-honest model (secure dot product, comparison, and set intersection), we first provided ways for developing them in the malicious model with the help of threshold decryption and zero knowledge proofs. Then we showed that these algorithms can further be improved in terms of efficiency by specialising them in the malicious model. As an example, we provided an efficient algorithm for secure dot product in the malicious model. We evaluated the performance of the algorithms in the malicious model by comparing them with the ones in the semi-honest model. We also included the performance analysis results of the circuit evaluation versions as a benchmark; thus showing the superiority of our methods. Finally, we discussed usability and applicability of the primitive algorithms in the malicious model when they are used within larger applications such as privacy-preserving k-means clustering and association rule mining.

As future work, we intend to extend our protocols to handle more than two parties. We also plan to analyse the performance of the other privacy-preserving algorithms in the malicious model using subprotocols devised in this paper.

## Acknowledgements

Murat Kantarcioglu is partially supported by Air Force Office of Scientific Research under Grant No. FA9550-07-1-0041.

Onur Kardes is partially supported by the National Science Foundation under Grant No. CCR-0331584.

Both authors thank Rebecca Wright for useful discussions and comments.

**References**

- Boudot, F., Schoenmakers, B. and Traoré, J. (2001) 'A fair and efficient solution to the socialist millionaires' problem', *Discrete Applied Mathematics*, Vol. 111, Nos. 1–2, pp.23–36.
- Canetti, R. (2000) 'Security and composition of multi-party cryptographic protocols', *Journal of Cryptology*, Vol. 13, No. 1, pp.143–202.
- Cramer, R., Damgard, I. and Nielsen, J.B. (2000) 'Multi-party computation from threshold homomorphic encryption', Technical Report RS-00-14, Basic Research in Computer Science – BRICS, June.
- Cramer, R., Damgard, I. and Nielsen, J.B. (2001) 'Multi-party computation from threshold homomorphic encryption', *Lecture Notes in Computer Science*, Vol. 2045.
- Damgaard, I., Fitz, M., Kiltz, E., Nielsen, J.B. and Toft, T. (2006) 'Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation', *Proceedings of the Third Theory of Cryptography Conference, TCC 2006*, pp.285–304.
- Damgard, I., Jurik, M. and Nielsen, J. (2003) 'A generalization of Paillier's public-key system with applications to electronic voting', [www.daimi.au.dk/~ivan/GenPaillier\\_finaljour.ps](http://www.daimi.au.dk/~ivan/GenPaillier_finaljour.ps).
- Du, W. and Zhan, Z. (2002) 'Building decision tree classifier on private data', in C. Clifton and V. Estivill-Castro (Eds.) *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, Australian Computer Society, Maebashi City, Japan, 9 December, Vol. 14, pp.1–8.
- Freedman, M.J., Nissim, K. and Pinkas, B. (2004) 'Efficient private matching and set intersection', *Eurocrypt 2004*, International Association for Cryptologic Research (IACR), Interlaken, Switzerland, 2–6 May.
- Gilburd, B., Schuster, A. and Wolff, R. (2004) 'Privacy-preserving data mining on data grids in the presence of malicious participants', *Proceedings of HPDC04*, Honolulu, Hawaii, June.
- Goldreich, O. (2004) *The Foundations of Cryptography*, Chap. 7, General Cryptographic Protocols, Cambridge University Press, Vol. 2.
- Jagannathan, G. and Wright, R.N. (2005) 'Privacy-preserving distributed k-means clustering over arbitrarily partitioned data', *Proceedings of the 2005 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Chicago, Illinois, 21–24 August, pp.593–599.
- Kantarcioğlu, M. and Clifton, C. (2004a) 'Privacy-preserving distributed mining of association rules on horizontally partitioned data', *IEEE TKDE*, September, Vol. 16, No. 9, pp.1026–1037.
- Kantarcioğlu, M. and Clifton, C. (2004b) 'Privately computing a distributed k-nn classifier', in J-F. Boulicaut, F. Esposito, F. Giannotti and D. Pedreschi (Eds.) *PKDD2004: 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Pisa, Italy, 20–24 September, pp.279–290.
- Kardes, O., Ryger, R.S., Wright, R.N. and Feigenbaum, J. (2005) 'Implementing privacy-preserving Bayesian-net discovery for vertically partitioned data', *ICDM Workshop on Privacy and Security Aspects of Data Mining*.
- Kissner, L. and Song, D. (2005) 'Privacy-preserving set operations', *Advances in Cryptology – CRYPTO 2005*.
- Lin, X., Clifton, C. and Zhu, M. (2005) 'Privacy preserving clustering with distributed EM mixture modeling', *Knowledge and Information Systems*, July, Vol. 8, No. 1, pp.68–81.
- Lindell, Y. and Pinkas, B. (2000) 'Privacy preserving data mining', *Advances in Cryptology – CRYPTO 2000*, Springer-Verlag, 20–24 August, pp.36–54.
- Lindell, Y. and Pinkas, B. (2002) 'Privacy preserving data mining', *Journal of Cryptology*, Vol. 15, No. 3, pp.177–206.
- Malkhi, D., Nisan, N., Pinkas, B. and Sella, Y. (2004) 'Fairplay – a secure two-party computation system', *Proceedings of the 13th Conference on Usenix Security Symposium*, Vol. 13, p.20.

- Paillier, P. (1999) 'Public-key cryptosystems based on composite degree residuosity classes', *EUROCRYPT*, pp.223–238.
- Vaidya, J. and Clifton, C. (2002) 'Privacy preserving association rule mining in vertically partitioned data', *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, 23–26 July, pp.639–644.
- Vaidya, J. and Clifton, C. (2003) 'Privacy-preserving k-means clustering over vertically partitioned data', *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, 24–27 August.
- Yang, Z. and Wright, R.N. (2006) 'Privacy-preserving computation of Bayesian networks on vertically partitioned data', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 9, pp.1253–1264.

## Notes

- 1 Since the equality protocol defined here is symmetric, we assume that  $P_0$  is corrupted without loss of generality.
- 2 The zero-knowledge proofs used in our protocols can be run in parallel. Since simulating the sequential execution is conceptually simpler, we assume that the zero-knowledge proofs are executed sequentially.
- 3 This is possible due to the special properties of the zero-knowledge proofs we have used. Please see Cramer *et al.* (2001) for details.

## Appendix

### Security proofs

In this section, we provide security proofs of our algorithms.

#### Security in the malicious model

In order to provide proofs of security, we must first give an exact definition of security. Since this is a well studied subject in SMC domain, we directly follow the security definitions given in the literature (Goldreich, 2004; Canetti, 2000; Cramer *et al.*, 2001). In our analysis, we focus on two-party protocols and assume that a malicious (*i.e.*, active) adversary can only corrupt one (and the same) party during the entire protocol. In other words, we only deal with active, static adversaries. As mentioned in Section 1, we focus on only two-party protocols. Therefore, we modify the definitions given in Cramer *et al.* (2001) for two-party case. Since we use the same zero-knowledge techniques given in Cramer *et al.* (2001); we follow the definitions from Cramer *et al.* (2001) instead of the similar ones given in Goldreich (2004) and Canetti (2000). We first introduce the execution of a protocol in the real model and then define the execution of a protocol in the ideal model. Finally, we define the security as emulating the real execution of a protocol in the ideal model.

*Definition A.1 The real model (Cramer et al., 2001)*

Let  $\pi$  be a two-party protocol where each party  $P_i$  has a secret input  $x_i^s$  and a public input  $x_i^p$ . Each  $P_i$  returns a private output  $y_i^s$  and a public output  $y_i^p$  after the execution of the protocol. Let  $A$  be an adversary that can corrupt any one (and only one) party during the execution of the protocol. Let  $\bar{x} = (x_1^s, x_1^p, x_2^s, x_2^p)$  be the participating parties input, let  $\bar{r} = (r_1, r_2, r_A)$  be the random inputs of the parties and the adversary, let  $C \in \{1, 2\}$  be the index of the corrupted party, and let  $z \in \{0, 1\}^*$  be the auxiliary input (*i.e.*,  $z$  could be seen as the prior information that can be used by the adversary.) Let  $k$  be the security parameter (*i.e.*,  $k$  could be seen as the parameter defining encryption key sizes.) We denote the output of the adversary after the execution of the protocol as  $ADV R_{\pi, A}(k, \bar{x}, C, z, \bar{r})$  and similarly the output of the party  $P_i$  as  $EXEC_{\pi, A}(k, \bar{x}, C, z, \bar{r})_i$ .

Let:

$$\begin{aligned} & EXEC_{\pi, A}(k, \bar{x}, C, z, \bar{r}) \\ &= (ADV R_{\pi, A}(k, \bar{x}, C, z, \bar{r}), \\ & \quad EXEC_{\pi, A}(k, \bar{x}, C, z, \bar{r})_1, \\ & \quad EXEC_{\pi, A}(k, \bar{x}, C, z, \bar{r})_2). \end{aligned}$$

We define the  $EXEC_{\pi, A}(k, \bar{x}, C, z)$  to be the random variable for a uniformly chosen  $\bar{r}$ . Finally, we define a distribution ensemble  $EXEC_{\pi, A}$  with security parameter  $k$  which is indexed by  $(\bar{x}, C, z)$  as:

$$EXEC_{\pi, A}(k, \bar{x}, C, z)_{k \in \mathbb{N}, \bar{x} \in \{0, 1\}^*, C \in \{1, 2\}, z \in \{0, 1\}^*}$$

*Definition A.2* *Ideal model (Cramer et al., 2001)*

Let  $f: N \times (\{0,1\}^*)^4 \times \{0,1\} \mapsto (\{0,1\}^*)^4$  be a probabilistic two-party function computable in probabilistic polynomial time. We define the output of  $f$  as:

$$f(k, x_1^s, x_1^p, x_2^s, x_2^p, r) = (y_1^s, y_1^p, y_2^s, y_2^p)$$

where  $k$  is the security parameter and  $r$  is the random input. In the ideal model, parties send their inputs to an incorruptible trusted party which draws  $r$  uniformly random, computes  $f$  and returns the party  $P_i$  its output value  $(y_i^s, y_i^p)$ . At the beginning of the execution the ideal model adversary  $S$  sees the  $x_1^p$  and  $x_2^p$  values and the secret  $x_C^s$  value for the corrupted party. After this point,  $S$  replaces  $x_C^s, x_C^p$  with the  $\bar{x}_C^s, \bar{x}_C^p$  values of its choice.  $f$  is then evaluated by the trusted party using the modified inputs. After the evaluation,  $P_i$  receives its output value  $(y_i^s, y_i^p)$ . Again adversary sees the  $y_1^p, y_2^p$  values and the  $y_C^s$  value for the corrupted party. Similar to the real model, let:

$$\begin{aligned} & IDEAL_{f,S}(k, \bar{x}, C, z, \bar{r}) \\ &= (ADV R_{f,S}(k, \bar{x}, C, z, \bar{r}), \\ & IDEAL_{f,S}(k, \bar{x}, C, z, \bar{r})_1, \\ & IDEAL_{f,S}(k, \bar{x}, C, z, \bar{r})_2) \end{aligned}$$

denote the collection of the outputs and  $IDEAL_{f,S}$  be the distribution ensemble indexed by  $(\bar{x}, C, z)$ .

*Definition A.3* *Security in the static malicious adversary setting (Cramer et al., 2001)*

Let  $f$  be a two-party function and let  $\pi$  be a protocol for two parties. We say that  $\pi$  securely evaluates  $f$  in the static setting if for any probabilistic polynomial time adversary  $A$ , there exists an ideal-model adversary  $S$  whose running time is polynomial in the running time of  $A$ , and such that:

$$IDEAL_{f,S} \stackrel{c}{\approx} EXEC_{\pi,A} \tag{1}$$

where  $\stackrel{c}{\approx}$  denotes the computational indistinguishability between two ensembles.

Security in this model implies that any adversary in the real-life model can be emulated by an adversary in the ideal model. The basic advantage of this simulation/emulation paradigm is that we can show that anything learned by the real-life adversary during the protocol execution is computationally indistinguishable from what is learned by an ideal model adversary. Since in the ideal model, any adversary can learn at most the final result and what is implied by the final result, proving that the real-life model adversary could be simulated by an ideal model adversary implies that real-life adversary could not learn anything more than the ideal model adversary. In other words, the real protocol execution reveal no more information to an adversary than what is revealed to an ideal model adversary.

Therefore, in the security proofs, we define an ideal model adversary  $S$  that runs any given real-life adversary  $A$  as a subroutine in a black-box fashion. We show that their views are computationally indistinguishable. We would like to stress that we only consider computational security in this paper. For protocols that are unconditionally secure, please refer to Damgaard *et al.* (2006).

We also need to combine several secure function evaluations to create new protocols. In order to prove that the composed protocol is secure, we first show that the protocol is secure given a trusted party (*i.e.*, oracle) that implements the functions used as a subroutine for the composed protocol. We name the function calls that use the given trusted party as oracle calls. Later, using the theorems stated in Canetti (2000), we can replace the oracle calls with secure protocols without violating security. In order to formalise the above intuition, we first define the hybrid model:

*Definition A.4* The hybrid model: two-party case (Cramer *et al.*, 2001)

The execution of the protocol  $\pi$  in the  $(h_1, h_2, \dots, h_m)$ -hybrid model proceeds as in the real model, except that the parties have oracle access to a trusted party  $T$  for evaluating the two-party functions  $h_1, \dots, h_m$ . These function evaluations proceed as in the ideal model. Similar to the previous definitions, we denote the output of the protocol with the following distribution ensemble:

$$EXEC_{\pi, A}^{h_1, \dots, h_m}.$$

Similar to the security definition given above, we define the security in the hybrid model by requiring that for any adversary operating in the hybrid model, there exists an adversary  $S$  in the ideal model such that:

$$IDEAL_{f, S}^c \approx EXEC_{\pi, A}^{h_1, \dots, h_m}. \quad (2)$$

As mentioned before, we can replace the oracle calls to function  $h_i$  with its secure implementation in the real model without sacrificing security. Please refer to Malkhi *et al.* (2004) for details.

*Theorem A.5* Protocol 3.1 is secure in the (decryption)-hybrid model assuming that the non-interactive zero-knowledge protocols used are secure in the malicious model.

*Proof*

In order to prove the security of the protocol, for any adversary  $A$  operating in the hybrid model, we need to define an adversary  $S_A$  operating in the real model such that the views of the both adversaries are computationally indistinguishable. In order to define such  $S_A$ , we will use  $A$  as a subroutine. Before the simulation starts,  $S_A$  will be given the description of the  $A$ , private input of the corrupted party  $X_0$ ,<sup>1</sup> the final result of the equality test  $b$ , public key  $pk$  and the private key of the  $P_1$ . Now we can define the  $S_A$  as follows:

- run  $A$  to get  $E_{pk}(x_0)$  along with the  $POK(e_{x_0})$

- run the simulator  $S_{POK}$  by giving the current state of the  $A$  and  $E_{pk}(x_0)$  as an input to  $S_{POK}$ . If the simulator of the proof fails then terminate the protocol, else set the state of  $A$  returned by the  $S_{POK}$
- if  $b$  is 1 then feed  $A$  with  $E_{pk}(x_0)$  else feed  $A$  with  $E_{pk}(r_a)$  for some random  $r_a \neq X_0$  along with the correct zero-knowledge proof. Let  $x_{S_A}$  be the plaintext value given to  $A$  in this step
- run  $S_{POCM}$  to simulate the zero-knowledge proof and set the state of  $A$  by the state returned by the  $S_{POCM}$ . If the proof fails then terminate. Also feed the  $A$  with the correct zero-knowledge proof for the encrypted value given to  $A$  in the previous state
- get the  $e_{(x_0 - x_{S_A})(r_0 + r_1)}$  for the oracle call to *Decrypt* function and give  $A$  a random number if  $b = 0$  else give 0 to  $A$
- output whatever  $A$  outputs.

We now need to prove that the view of  $S_A$  is computationally indistinguishable from the execution in the hybrid model. First note that until Step 2, the view of the  $A$  in the simulation is statistically indistinguishable to the view of  $A$  in the hybrid model. Security of the zero-knowledge guarantees that on computationally indistinguishable inputs, the output state of the zero-knowledge proof simulator is identical to state of  $A$  in the hybrid protocol. With the similar arguments, we can argue that the state of  $A$  before the Step 5 is identical to the state in the hybrid model. Now we need to show that the result returned by the decryption call in the simulation is statistically indistinguishable from the one seen by the  $A$  in the actual implementation. If  $b = 1$ , then in the both executions,  $A$  will be given 0, if  $b = 0$  than in the simulation,  $A$  can see any value with equal probability, in the hybrid model execution  $A$  will get  $(x_0 - x_1).(r_0 + r_1)$ . Since  $r_1$  is random, the probability that  $(x_0 - x_1).(r_0 + r_1)$  equals to zero is negligibly small in terms of the security parameter. Also  $(x_0 - x_1).(r_0 + r_1)$  is distributed uniformly in  $Z_n^*$  since all operation are done modulo  $n$ . Therefore, the state of  $A$  after the Step 5 is the same in both executions. This concludes our proof.

*Theorem A.6* Protocol 3.2 is secure in (private decrypt)-hybrid model assuming that the non-interactive zero-knowledge protocols used are secure in the malicious model.

*Proof*

Again, for any adversary  $A$  operating in the hybrid model, we need to find an adversary  $S_A$  operating in the ideal model. In order to simplify our simulator, let us describe  $S_A$  for two different cases depending on whether  $P_0$  or  $P_1$  is corrupted. First let us assume that  $A$  controls  $P_0$ , we can define  $S_A$  as follows:

- $S_A$  gets the final result  $\sum_i^n (x_{0i} \cdot x_{1i}) + r_1$  as input.
- $S_A$  uses the simulator  $S_{POK}$  for each  $x_{0i}$  input sequentially.<sup>2</sup> If any one of the proofs terminate then  $S_A$  terminates also.

- $S_A$  sets the state of  $A$  returned by the last run of  $S_{POK}$ .
- $S_A$  simulates the honest  $P_1$  by constructing the required correct zero knowledge proofs and feeds  $A$  with those proofs.
- $S_A$  simulates the joint call to private decrypt function by returning  $\sum_i^n (x_{0i} \cdot x_{1i}) + r_1$  to  $A$ .
- $S_A$  outputs whatever  $A$  outputs.

Please note that the state of the  $A$  after the last execution of  $S_{POK}$  is identical in both worlds. Since zero knowledge proofs seen by the  $A$  that are given by  $S_A$  for simulating the correct behaviour of  $P_1$  are encrypted using a semantically secure encryption, the view of  $A$  in both worlds should be computationally indistinguishable. Therefore, the state of  $A$  before the oracle call to private decrypt function should be identical. Since  $S_A$  gives the correct result to  $A$ , the outputs in both worlds should be computationally indistinguishable.

For the case where  $P_1$  is corrupted, the construction of  $S_A$  is very similar to the case where  $P_0$  is corrupted. Only the order of execution of simulators  $S_{POK}$  and  $S_{POCM}$  are different. We omit the further details here.

*Theorem A.7* *The Protocol 3.3 is secure in (secure equality, private decrypt) -hybrid model assuming that the non-interactive zero-knowledge protocols used are secure in the malicious model.*

*Proof*

Without loss of generality, let us assume that real-world adversary  $A$  controls  $P_0$ . Again for any real adversary  $A$ , we define a simulator  $S_A$  such that output of the adversary is computationally indistinguishable in both worlds. Again, let us assume that  $S_A$  is given the output of the protocol. Now we can define the  $S_A$  as follows:

- run  $S_{POK}$  to verify the zero knowledge proofs and set  $A$  to the state returned by  $S_{POK}$
- use the  $S_{POK}$  to learn the  $x_{0i}$  values<sup>3</sup> with overwhelming probability. If  $S_{POK}$  does not return the  $x_{0i}$  values then abort
- generate random  $x_{1i}$  and  $r_1$  values such that the dot product result is consistent with the one given to  $S_A$
- feed  $A$  with the correct zero knowledge proofs for all  $E_{pk}(x_{1i})$  and  $E_{pk}(r_1)$
- calculate the correct encrypted value  $e_{r_0}^1$
- get the input from the  $A$  for secure equality protocol
- run the simulator for the secure equality protocol with the correct inputs
- set the state of  $A$  to the state returned by the secure equality protocol simulator
- if the both inputs are equal then return the correct result to  $A$  after the private decrypt call else abort

- output whatever  $A$  outputs.

We need to show that the output of the  $A$  in the both worlds should be computationally indistinguishable. First note that, since  $A$  only sees the encrypted  $x_1$  values, the state of  $A$  after Step 5 is identical, otherwise  $A$  could be used as distinguisher for the homomorphic encryption. Since the inputs to the simulator for secure equality are computationally indistinguishable in both worlds, the state of  $A$  after the Step 8 is identical. Finally before the last step, the input given to  $A$  in the both worlds are the same and states are identical.

Therefore, the output of  $A$  in both worlds should be computationally indistinguishable. Again the simulator for the case where  $A$  controls  $P_1$  is the same. Therefore, we omit the discussion of that case.

*Theorem A.8*    *The Protocol 3.4 is secure in (threshold decryption)-hybrid model assuming that the non-interactive zero-knowledge protocols used are secure in the malicious model.*

*Proof*

Again, we need to prove that for any given adversary  $A$  operating in the hybrid model, we can simulate its actions in the ideal world. The  $S_A$  operating in the ideal world, is very similar to the one given in the proof of the Theorem A.6. First let us assume that  $A$  controls  $P_0$ , we can define  $S_A$  as follows:

- get the final results for all  $I_i$  values
- use the simulator  $S_{POK}$  for each  $x_{0i}$  input sequentially. If any one of the proofs terminate then  $S_A$  terminates also
- set the state of  $A$  returned by the last run of  $S_{POK}$
- simulate the honest  $P_1$  by constructing the correct zero knowledge proofs that are required. Feed  $A$  with those proofs
- simulate the joint call to private decrypt function by returning the correct  $I_i$  values for all  $i$
- Output whatever  $A$  outputs.

Please note that the state of the  $A$  after the last execution of  $S_{POK}$  is identical in both worlds. Since zero knowledge proofs seen by the  $A$  that are given by  $S_A$  for simulating the correct behaviour of  $P_1$  are encrypted using a semantically secure encryption, the view of  $A$  in both worlds should be computationally indistinguishable. Therefore, the state of  $A$  before the oracle calls to private decrypt function should be identical. Since  $S_A$  gives the correct result to  $A$ , the outputs in both worlds should be computationally indistinguishable.

Since the case where  $A$  controls the  $PI$  is similar, we omit the simulator for that case.