UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Secure Multiparty Computation

propusă de

Lucian-Dan Neștian

Sesiunea: Iulie, 2019

Coordonator științific

Lect. Dr. Sorin Iftene

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Secure Multiparty Computation

Lucian-Dan Neștian

Sesiunea: Iulie, 2019

Coordonator științific

Lect. Dr. Sorin Iftene

Avizat,

Îndrumător lucrare de licență,

Lect. Dr. Sorin Iftene.

Data: .... Semnătura: ....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Neștian Lucian-Dan** domiciliat în **România, jud. Vaslui, mun. Vaslui, str. Castanilor, nr. 5, bl. C1, et. 1, ap. 27**, născut la data de **24 octombrie 1997**, identificat prin CNP **1971024374511**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2019, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Secure Multiparty Computation** elaborată sub îndrumarea domnului **Lect. Dr. Sorin Iftene**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: .... Semnătura: ....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Secure Multi-party Computation**, codul sursă al programelor și celelalte conținuturi (grafice, multi-media, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea ”Alexandru-Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Lucian-Dan Neștian**

Data: .... Semnătura: ....

Contents

Introduction **2**

1 Preliminaries **4**

1.1 Linear algebra 4

1.1.1 Groups and Rings 4

1.1.2 Fields and homomorphisms 5

1.2 Random Variables 5

1.3 Secret sharing 6

1.4 Oblivious Transfer 7

1.5 Homomorphic Encryption 10

2 Secure Multiparty Computation **11**

2.1 Abstract 11

2.2 Multiparty Computation 11

2.3 Secure addition and its applications 12

2.4 Secure multiplication and Match-Making 15

2.5 Yao's millionaire problem 17

2.5.1 Garbled Circuits 17

2.5.2 Secure Two-Party Protocols for Semi-Honest Adversaries 21

2.5.3 The garbled circuit construction 23

2.5.4 Yao's Two-Party Protocol 24

2.6 The multiparty case 26

2.6.1 What if parties do not follow instructions? 26

2.6.2 Adversarial Power 28

3 Application **31**

3.1 Data mining 31

3.1.1 KMeans algorithm 31
3.1.2 Privacy-preserving clustering over vertically partitioned data . . 32
3.1.3 Implementation 42

Conclusions **46**

Bibliography **46**

Introduction

In an information-driven society, the day-to-day life of individuals and organizations is full of cases where all kinds of private data represents an important resource. For an individual, this information may be related to his economic status, such as his income, his current credits, various tax data, or information related to his health, such as risks, medication usage, etc. For an organization, this private data might concern its business status, namely profit analyses, reports, etc. The benefit of having access to all these kinds of data comes mainly from that fact that one can compute on the data, in hopes of tuning it into something beneficial. But, handling private information distributed on multiple entities can be a challenge if it is in the context of preserving the privacy.

Modern techniques within the field of cryptography do have the potential to solve this problem. In particular, one research area is devoted to secure multiparty computation (SMC). SMC is a distributed computational model, in which several parties try to compute a common function that requires everyone's information, while keeping their own information private. SMC protocols are considered privacy-preserving meaning that they leak nothing about parties' private data other that what can be deducted from the function's output. SMC protocols started to arise with the introductions of Yao's millionaire problem [Yao82] in 1982. In 1987 a milestone was reached when there was demonstrated that any computation can be done in SMC context [GMW87]. Since then, the potential of this technique has grown significantly.

The full benefits of SMC protocols are related with the advancements in trends such as internet of things (IoT), big data and cloud computing which require a more refined control over information than what can be currently achieved with classic encryption. In some sense, SMC can be compared with the discovery of public key encryption in 1970s; it started out as a theoretical result but today it is widely used for security.

0.1 Outline

This thesis gives a detailed view over the construction and implementation of SMC protocols. It provides the necessary background in linear algebra and secret-sharing techniques to introduce some SMC primitives, namely oblivious transfer,garbled circuits and homomorphic encryption in chapter 2. This chapter also familiarizes the reader with the fundamental properties that a protocol needs to satisfy in order to be secure in the context of SMC, while also describing the types of possible adversarial behaviours. A privacy-preserving clustering algorithm is presented in chapter 3 as well as its related security discussion. Also, the implementation in Python of said protocol is summarized in this chapter. Finally, the conclusions are made in chapter 3.1.3.

0.2 Contributions

Our main contribution is to present the state of art of the work made in the field of secure multiparty computation. Along each primitive, comprehensively explained examples are present in order to make things clear. Another main contribution represents the implementation for the problem of privacy-preserving clustering with k-means algorithm, described in [VC03]. While there are some solutions that address this type of problem ([Bog+12], [GC16]), our implementation is different because it is set on vertically partitioned data. Clustering with the privacy property in mind is still an open problem and it will be considered for the future research.

Chapter 1

Preliminaries

1.1 Linear algebra

In this section we introduce some basic notions used throughout the paper. The other main purpose of this chapter is to establish our notation.

1.1.1 Groups and Rings

Definition 1.1.1. A group is a tuple $(G, \bullet)$ where G is a set and $\bullet$ is an operation that combines two elements from G. To qualify as a group, $(G, \bullet)$ must satisfy the following group axioms[Ţip06]:

- **Closure**

  $\forall a, b \in G$, $a \bullet b \in G$

- **Associativity**

  $\forall a, b, c \in G$, $(a \bullet b) \bullet c) = a \bullet (b \bullet c)$

- **Identity element**

  There is an element $e \in G$ such that $\forall a \in G, a \bullet e = e \bullet a = e$. This element is unique.

- **Inverse element**

  $\forall a \in G, \exists b = a^{-1}$ such that $a \bullet b = e$. If the operation $\bullet$ is *commutative* ($a \bullet b = b \bullet a$) the group is called an *abelian group*.

Definition 1.1.2. As presented in [AM69], a ring $(A, +, \bullet)$ is a set with two binary operations such that:

1. $(A, +)$ is an *abelian group*.

2. The $\bullet$ operation is $distributive$ over the $+$ operation:

   $\forall a, b, c \in A, (a \bullet (b + c) = a \bullet b + a \bullet c), ((a + b) \bullet c = a \bullet c + b \bullet c)$

3. $\forall a, b \in A, a \bullet b = b \bullet a$

4. There is an element $1 \in A$ such that $\forall a \in A, a \bullet 1 = 1 \bullet a = a$

We will refer to $+$ operation as $addition$ and $\bullet$ operation as $multiplication$.

1.1.2 Fields and homomorphisms

Definition 1.1.3. A field $(F, +, \bullet)$ is a set with two binary operations such that:

1. $(F, +, \bullet)$ is a *ring*

2. $(F, \bullet)$ is an *abelian group* with respect to multiplication, having $e = 1$

   A field is called $finite$ if it contains a finite number of elements.

   The characteristic of a field p is the smallest prime natural number such that $p \bullet 1 = 0$, where

   $n \bullet a = \underbrace{a + a +, \ldots, +a}_{n \text{ times}}, \forall a \in F.$

   We will next define a homomorphism as in [Ţip06]:

Definition 1.1.4. A $homomorphism$ between two algebric structures of the same type is a map that preserves the operations of the structures: $f : A \to B$, $\bullet$ - binary operation, then:

$$f(x \bullet y) = f(x) \bullet f(y), \forall x, y \in A$$

and

$$f(x + y) = f(x) + f(y), \forall x, y \in A$$

1.2 Random Variables

Definition 1.2.1. [CDN15]

The $range$ of a random variable X is the set of elements x for which the random variable X outputs x with a probability greater than 0.

From now on, a *random variable with finite range* will be referred to as a random variable for simplicity.

For a random variable X we will use $Pr[X = x]$ to denote the probability of X outputting x. When given X there exists a finite set D such that $\sum_{x \in D} Pr[X = x] = 1$.

Definition 1.2.2. *Negligible function*

A function f is called negligible if for every polinomyal p there exists an integer N such that for all $n > N$ it holds that $f(n) < 1/p(n)$.

Definition 1.2.3. *Computational indistinguishablility* [LP09]

Let X and Y be two random variables. These random variables are called computationally indistinguishable (denoted $X \stackrel{c}{\equiv} Y$) if for every polynomial-time distinguisher Ds there exists a function f that is negligible in n such that:

$$[Pr[Ds(X) = 1] - Pr[Ds(Y) = 1]] < f(n) \tag{1.1}$$

Note that an event that has a negligible probability happens so infrequently that we can effectively discard it.

Definition 1.2.4. *Statistical Distance*

Let X_0 and X_1 be two random variables with range D. Then:

$$\delta(X_0, X_1) = \frac{1}{2} \sum_{d \in D} |Pr[X_0 = d] - Pr[X_1 = d]| \tag{1.2}$$

is called the statistical difference between X_0 and X_1.

1.3 Secret sharing

The main tool to build a protocol with passive security are so-called secret-sharing schemes. The theory on secret scheme is a vast and an interesting field on its own, with many applications on secure multiparty computations. In this section we concentrate on a particular scheme, namely Shamir's secret sharing scheme [Sha79].

Based on polynomials over a finite field $\mathbb{F}$, we will set, for simplicity, $\mathbb{F} = \mathbb{Z}_p$ with $|\mathbb{F}| > n$ and some prime $p > n$.

As said in [CDN15], a value $s \in \mathbb{F}$ is *shared* by choosing a random polynomial, denoted $f_s(X) \in \mathbb{F}[X]$, of degree at most t such that $f_s(0) = s$. In the sharing part of a protocol, a party P_j privately receives the share $s_j = f_s(j)$. The main idea of this method is that any set of t or fewer shares contain no information on s, while $t+1$ or more shares can be used together to reconstruct the secret s. Both of these properties are proved using Lagrange interpolation.

Lagrange Interpolation

Given $h(X)$, a polynomial over $\mathbb{F}$ of degree at most l and C, a subset of $\mathbb{F}$ such that $|C| = l+1$, then:

$$h(x) = \sum_{i \in C} h(i)\delta_i(X)$$

where $\delta_i(X)$ is the degree l of polynomial such that, $\forall i, j \in C, \delta_i(j) = 0$ if $i \neq j$ and $\delta_i(j) = 1$ if $i = j$. More exactly,

$$\delta_i(x) = \prod_{j \in C, j \neq i} \frac{X-j}{i-j}$$

Since $\delta_i(X)$ is the product of l polynomials, it is a polynomial of degree at most l. Therefore, the right side of the polynomial

$$\sum_{i \in C} h(i)\delta_i(X)$$

is a polynomial of degree at most l and that on input i evaluates $h(i)$ for $i \in C$. Moreover, $h(x) - \sum_{i \in C} h(i)\delta_i(X)$ is 0 across all points in C.

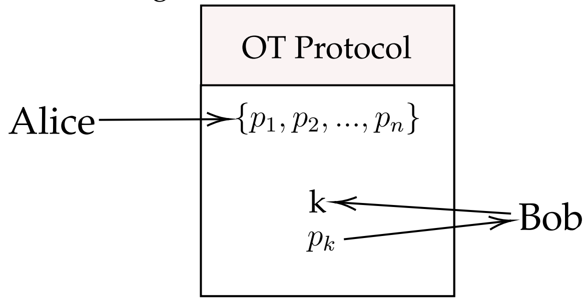
1.4 Oblivious Transfer

Firstly described in [EGL85], the general *Oblivious Transfer* protocol allows two participants to share a value p_k among a set of values $\{p_1, p_2, ..., p_n\}$ such that one party doesn't know which value the other has chosen.

The figure 1.1 encapsulates the flow of this protocol, not the protocol itself. The arrows denote information accessible by different participants.

We will approach the *1-out-of-2-oblivious-transfer (OT)*, which is most commonly used. The protocol requires two participants, denoted as Alice and Bob, who would like to share one out of the two possible values, $\{P_1, P_2\}$. Presume that Bob chooses the x^{th} value, $x \in \{0, 1\}$. Then, the following proprieties must be satisfied:

Figure 1.1: OT overview



- Alice should not know the value of x
- Bob should not know more than the value he requested, x

The following steps of the protocol were selected from Lindell's talk [LP09] and [EGL85]. Keep in mind that the security propriety is only assured in the presence of *semi-honest* adversaries. Also, when considering semi-honest adversaries, the general case (*k-out-of-n*) can be obtained by running these steps in parallel many times.

Protocol 1 1-out-of-2 OT

Prerequisites. Alice has a RSA [RSA78] key-pair (N, e, d) and Bob has the public part of it, (N, e).

Inputs. Alice has a set of values $\{p_0, p_1\}$, $p_0, p_1 \in \{0, 1\}$. Bob has $k \in \{0, 1\}$.

Goal. Bob obtains p_k in a private manner, without accessing anything other value than p_k.

The protocol:

1. Alice generates two values at random, $\{x_0, x_1\}$ associated with $\{p_0, p_1\}$.
2. Bob generates a random r, computes the encryption of r blind with x_k and sends it to Alice: $v = (x_k + r^e) \bmod N$.
3. Alice receives r and now knows that one of these values will be equal to r: $r_0 = (v - x_0)^d \bmod N$ and $r_1 = (v - x_1)^d \bmod N$, but she doesn't know which.
4. Alice sends to Bob $\{p'_0, p'_1\}$ with $p'_0 = p_0 + r_0$, $p'_1 = p_1 + r_1$.
5. Bob decrypts the p'_k since he knows which x_k he selected earlier and learns $p_k = p'_k - v$.

As noted in [Has+19] , from a theoretical standpoint, MPC protocols can be built using Oblivious Transfer alone [Has+19] [Rab05] [Kil88] but the main breakthrough feature that makes Oblivious Transfer suitable for building efficient MPC protocols is that Oblivious Transfer is equivalent to a seemingly weaker protocol denoted as *Random Oblivious Transfer (ROT)* [Cré87], where the bit choice k is not provided as an input, but rather it is randomly generated by the protocol. The output consists in two related pairs of bits, (p_0, p_1) and (p_k, k), where p_0, p_1, k are uniformly chosen at random from the set $\{0, 1\}$. Given a random correlation (the pairs (p_0, p_1) and (p_k,k)), Alice and Bob can replicate the Oblivious Transfer behaviour using only three bits of communication.

Considering the fact that ROT implies OT, parties can do all the necessary ROT computations needed for a particular protocol in advance, in an input-independent $offline$ phase. After this pre-processing phase, they consume these pre-generated OTs in a so-called *online* phase and execute the protocol with minimal communication costs and without expensive public-key operations. This separation between online and offline phases makes the online phase of the protocol very efficient, but the problem of keeping the pre-processing phase equitable in terms of computations. There exist two fundamentally different approaches that deal with handling the offline phase, either through a *trusted dealer* or a *cryptographic batched correlation-generation protocol*.

In the trusted dealer model, a semi-trusted dealer distributes correlated randomness to all other parties. The trusted dealer has no input and doesn't have access to private information hence the dealer only needs to be trusted to generate and distribute random values to the appropriate parties. In presence of such dealer, the offline phase of some MPC becomes extremely efficient.

Without a trusted party, OTs can be generated efficiently through some *Oblivious transfer extensions*. In such protocols, a small number (denoted *seed* or *base*) of OTs are converted into a large number of ROTs through the use of efficient symmetric-key primitives (i.e. AES) [Ish+03].

1.5 Homomorphic Encryption

Definition 1.5.1. A homomorphic encryption scheme is a triple (G, E, D) where:

- G is a polynomial time algorithm used to generate the public/private key-pair.
- E is the encryption algorithm.
- D is the decryption algorithm.

A homomorphic encryption scheme has the following property:

$$E(m_1) \bullet E(m_2) = E(m_1 \bullet m_2)$$

Homomorphic encryption schemes are an important constituent of secure multiparty protocols. Some of the schemes developed through the course of the years include additively homomorphic encryption [Pai99a], multiplicatively homomorphic encyption [Gam84] and fully homomorphic encryption [LNV11].

Chapter 2

Secure Multiparty Computation

2.1 Abstract

We are in a situation in which a number of parties would like to compute a predefined set of operations that require the private data set of each party. The parties want to learn some of the output as well as keeping their respective inputs private. A trivial solution would be to find some party T that is trusted by every party. All private inputs are send to T, T performs the necessary computation and broadcasts the output to every party involved. Then, T $deletes$ all private data which he had. At first sight, this approach seems to solve our initial problem. The main issue with this solution is that of finding a T such that $everyone$ trusts. In practice, this is rarely the case. How would the parties agree on trusting such entity if they do not trust each other in the first place? Furthermore, even if they did find such T we would face the fact that T is now a *single point of failure*: if T fails to compute the output because of arbitrary reasons, the whole agreement is for nothing; if T is compromised, all the respective private data is leaked to everyone.

Now, our problem becomes computing the computations of private inputs without relying on a trusted party. We shall present some solutions for this problem.

2.2 Multiparty Computation

Let us think about our situation into more formal way. Each party P_i has its own private data that he can input, x_i. All parties $P_1, P_2 ... P_n$ have a set goal in mind from the start, that of computing y which is the result of a function f that takes n(number of parties) inputs: $y = f(x_1, x_2 ... x_n)$. This should only be possible when certain proprieties apply (according to [Lin09] and [CDN15]):

- *Correctness*: Each party P_i is guaranteed from that start that the output y that he receives is the correct value computed.
- *Privacy*: The only new information revealed to each party P_i is y. In most situations, the only information that should be learned about other parties' inputs is in direct relation with the output itself.

Having a protocol that computes f such that both $Privacy$ and $Correctness$ are ensured is referred to as computing f *securely*.

To establish an early intuition, we can discuss a particular case as presented in [CDN15]. As one example of how these proprieties can be achieved while solving a real world problem, one may think of P_i as a participant in an auction. Its private input, x_i, is a number which represents the amount P_i wishes to bid in said auction. The function $f(x_1, x_2 ... x_n)$ outputs a pair $(z, j) = y$ where $\exists x_j = z$ and $\forall i \neq j, i \in [1, n], x_i < x_j$ in which z represents the highest amount that has been bidden and j represents the identity of the winner.

The first thing we must establish in order to compute this function without relying on a said trusted party T is a $protocol$. A $protocol$ is a set of successive operations or instructions that each party has to follow in order to obtain the desired goal. As of now, we assume that each party follows these instructions thoughtfully and does not deviate from this protocol. Later on, we address the case in which a party decides to deviate from the protocol in order to obtain new information regarding the others' private inputs or to simply disturb the communication flow in section [2.6]. We also assume that parties can communicate securely, i.e., it is possible to send a message from party P_i to P_j such that no third party intercepts this message and that their communication channel is resilient.

2.3 Secure addition and its applications

Another specific problem, as mentioned in [CDN15], is that of securely computing $f(x_1, x_2 ... x_n) = \sum_{i=1}^{n} x_i$, where the private inputs x_i are real numbers. Finding a protocol that solves for this problem can have a large number of applications. For example, if we consider each number $x_i \in$ {0,1} representing a vote of type yes/no and each party P_i an entity that wants to express his vote privately, we now get a protocol for secret voting. Indeed, the output $f = \sum_{i=1}^{n} x_i$ denotes the number of yes votes, respectively, the number of no votes is $n - f$. Furthermore, if the computation is secure then the parties learn nothing about how a particular party voted.

Let us analyze an in-depth approach for this protocol where $n = 3$.

We choose p as a prime number and $\mathbb{Z}_p = \{0, 1...p-1\}$. Each party P_i holds a $share$ of the total sum s, say x_i. We will refer to x_i as a number in $\mathbb{Z}$. Other methods of *secret sharing* have been discussed in chapter 1.3. Like the name of the technique says, we aim to to provide each party P_i a way to spread information about its secret number x_i across all present parties, such that together they hold the full information about s_i, but, at the same time, no party alone (or formally, a group of less than t parties) can uncover x_i (except, of course, for P_i which holds the share).

Essentially, to $share\ a\ secret$ x_1 with the other parties, P_2 and P_3, P_1 has to choose two uniformly random numbers r_1, r_2 and sets

$$(r_3 = x_1 - r_1 - r_2) \bmod p.$$

This is another way of saying that P_1 chooses three random numbers from $\mathbb{Z}_p$ under the constraint that $x_1 = (r_1 + r_2 + r_3) \bmod p$. An easily observable property about these numbers is that all three of them are uniformly chosen from $\mathbb{Z}_p$ i.e. for each of them, all values of $\mathbb{Z}_p$ are equally likely to be chosen.

After this step of the protocol, P_1 has to send $privately$ both r_1 and r_3 to P_2, r_1, r_2 to P_3, and keeps r_2 and r_3 to himself.

We observe that at this stage of the protocol, some basic proprieties are satisfied: firstly, the secret x_1 is kept private in a sense that neither P_1 or P_2 knows anything about the secret. Secondly, x_1 can be reconstructed if the shares from at least two parties are used. Looking at these proprieties in the sense of secure multiparty computation:

- $Privacy$: Although P_1 has endorsed the sharing of information with the other parties using r_1, r_2 and r_3, both P_2 and P_3 can not determine x_1 with a meaningful probability. The reasoning for P_2 in particular is as follows: he has shares r_1 and r_3 and knows that $x_1 = r_1 + r_2 + r_3 \bmod p$. For this party, finding an $x \in \mathbb{Z}_p$ such that $x = x_1$ means that it now knows $r_2 = x - x_1 - x_3 \bmod p$. This is clearly a possibility. Recalling the fact that r_2 is chosen uniformly random in $\mathbb{Z}_p$ leads to acknowledging that all values are possible. Any other choice, say $x_1 = x_1' \neq x$ is also a possibility. If this was the case, the information that P_2 gains is that

$r_2' = x_1' - r_1 - r_3$ mod p which is a different value from r_2, but just as likely to be deducted. The conclusion that follows is that, from P_2's point of view, all values in $\mathbb{Z}_p$ can potentially be the initial secret x_1 and are all equally possible. Similarly, we conclude that P_3 can not determine x_1 in a reliable way.

- $Correctness$: If two parties share their information, the secret can be deduced because all its shares are known and one party can simply use the operation of addition under p.

Protocol 2 Secure Addition Protocol

Prerequisite. All parties agree on p - prime.

Inputs. For all $i \in \{1, 2, 3\}$, party P_i holds an input $x_i \in \mathbb{Z}$.

Goal. Parties securely compute the sum of their private inputs.

The protocol:

1. **Setup.**

   Each party P_i computes and distributes shares of his secret x_i as described above: it chooses r_{i_1}, r_{i_2} uniformly random from $\mathbb{Z}_p$ and sets $r_{i_3} = x_i - r_{i_1} - r_{i_2}$ mod p.

2. Each party P_i privately sends shares r_{i_2} and r_{i_3} to P_1, r_{i_1}, r_{i_3} to P_2 and r_{i_1} and r_{i_2} to P_3. Notice that using this rule, P_i sends some shares to itself, meaning that it keeps said shares. For instance, after this step, P_1 has access to r_{1_2}, r_{1_3}, r_{2_2}, r_{2_3}, r_{3_2} and r_{3_3}.

3. Each party P_i computes the addition of its shares in the following way: for each $j \neq i$, $s_j = r_{1_j} + r_{2_j} + r_{3_j}$ mod p. Then, he broadcasts the obtained values to all other parties.

4. All parties can now compute the sum $s = s_1 + s_2 + s_3$.

Before further analyzing the protocol for secure addition, we need to verify the $correctness$ property, i.e. the sum is indeed the correct one computed by each party. This results from:

$$s = \sum_j s_j \text{ mod } p = \sum_i \sum_j r_{i_j} \text{ mod } p = \sum_i x_i \text{ mod } p$$

Essentially, this means that no matter the secrets x_i or how they are fragmented into secret shares, the protocol computes the sum modulo p. If the parties agree on setting the meaning of an input $x_i = 1$ as a yes vote, $x_i = 0$ for a no vote and $p > 3$, p-prime, then $s = \sum_i x_i$ mod p = $\sum_i x_i$ is indeed the number of yes-votes and can be computed using this protocol.

2.4 Secure multiplication and Match-Making

Towards achieving a general secure computation, one shall need more than secure addition alone. Such goals are not easy to obtain. But, relying on reliable past results can help. Here, we will see how a new protocol that solves another problem can be constructed using the results from the previous section.

As described in [CDN15], suppose that there are given two numbers, $a, b \in \mathbb{Z}_p$ (that have been secret shared), we wish to securely compute the product ab mod p. As described in the above section, both a is equal to $a_1 + a_2 + a_3$ mod p, while b is equal to $b_1 + b_2 + b_3$ mod p. Then:

$$ab = a_1b_1 + a_1b_2 + a_1b_3 + a_2b_1 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3 \text{ mod } p$$

The first observation is that if the shares of a and b are distributed following the Secure addition protocol, for every product a_ib_j there is at least one party that has the shares a_i and b_j and can evaluate a_ib_j. For instance, party P_1 has been given a_2, a_3, b_2 and b_3 and can compute $a_2b_2, a_2b_3, a_3b_2, a_3b_3$ accordingly. Now, the situation calls for a protocol that can securely compute the sum of some numbers to determine the product ab, which is essentially what Secure addition protocol does. For the sake of consistency, we will keep the number of involved parties equal to 3. The construction for any n case is obtained similarly.

Protocol 3 Secure Multiplication

Prerequisite. All parties agree on p - prime.

Inputs. P_1 holds $a \in \mathbb{Z}_p$, P_2 holds $b \in \mathbb{Z}_p$ and P_3 has no input.

Goal. Parties securely compute the product ab.

The protocol:

1. **Setup.**

   P_1 distributes shares a_1, a_2, a_3 of a while P_2 distributes shares b_1, b_2, b_3 of b to all other involved parties.

2. P_1 locally computes $s_1 = a_2b_2 + a_2b_3 + a_3b_2 + a_3b_3 \bmod p$, P_2 computes $s_2 = a_3b_3 + a_1b_3 + a_3b_1 \bmod p$ and P_3 computes sum $s_3 = a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2 \bmod p$.

3. The parties then use the Secure addition protocol to securely compute the sum $s = s_1 + s_2 + s_3 \bmod p$, where each s_i is the input of party P_i, $i \in \{1, 2, 3\}$.

4. All parties can now compute the sum $ab = s_1 + s_2 + s_3$.

This protocol respects the $correctness$ property because how ab is constructed resulting in $ab = s_1 + s_2 + s_3 \bmod p$. To show that the $privacy$ property is preserved (i.e. nothing apart from $ab \bmod p$ is revealed) one notes that nothing new about a and b is in the $Setup$ step and because the Secure addition protocol is private, nothing except the sum of the inputs is revealed at the output, which is the sum equal to $ab \bmod p$.

A noticeable remark is that if, by convention, a and b are from set $\{0, 1\}$, secure multiplication has meaningful applications: consider two parties that would like to know if they trust one another. Say party P_1 wants to know if a collaboration with P_2 would be possible but without running the risk of disclosing the fact that they are interested in collaborating if P_2 does not feel the same way. The problem can be solved if party P_1 sets $a = 1$ if it trust party P_2 and $a = 0$ otherwise. In a similar way, P_2 chooses b to be either 0 or 1. Then, they securely compute function $f(a, b) = ab \bmod p$. The result is 1 if and only if there is mutual interest between the two parties involved. But on the other hand, if either of party chooses 0 as their input, they learn *nothing* *new* from the exchange. In real-world, companies can play the role of parties, and after participating at the protocol they can learn about other competitors' interest in making collaborations, without disclosing to the public their strategies.

The above argument assumes that both parties choose their inputs honestly (i.e. they pursue their real interest). In the section 2.6, we discuss what happens when parties do not follow the protocol and other implications of such behaviour.

2.5 Yao's millionaire problem

Yao's millionaire problem [Yao82] is a famous problem which was first stated in 1982 by Andrew Yao. Since then, different approaches of solving this problem have vastly widen the secure multiparty computation field, with protocols for solving it becoming more and more efficient: from the solution proposed by Yao using garbled circuits (as presented in [Yao82] and [IG03]), and an approach using homomorphic encryption [LT05].

The problem discusses two millionaires, in literature noted Alice and Bob, that want to find out which one of them is richer without revealing their actual wealth. In our multiparty computation environment, the problem corresponds to the case when $m = 2$, Alice and Bob are our parties P_1 and P_2 respectively, and the function which we would like to securely compute is $f(x_1, x_2) = 1$ if $x_1 < x_2$ or $f(x_1, x_2) = 0$, otherwise. The most obvious applications of such computation are in the context of e-commerce (ex: auction pricing) and data mining (ex: feature selection).

2.5.1 Garbled Circuits

Garbled circuits are a concept firstly introduced by Yao in order to express a $protocol$ for such computations. His result is central to the field of secure computation.

As presented in [Lin09] [LP09], having f, a polynomial-time function (assuming it is deterministic), and x_1, x_2 as parties inputs', we need to view f as a boolean circuit $C(x_1, x_2)$. To further examine how such protocol would work, we would need to describe how such a circuit is computed. Having x_1, x_2, $C(x_1, x_2)$ is computed gate-by-gate, from the $input$ wires to the $output$ wires. Once the gate g has received any values α and $\beta \in \{0, 1\}$, it can send through the outgoing wires the value $g(\alpha, \beta)$. The output of the circuit C is determined by the value obtained in the last output wires of it. Therefore, computing such circuit C means allocating values v_i to each wire w_i (with $v_i \in \{0, 1\}$, $i \leq n, n$ - number of wires of the circuit). There are four different types of wires as noted both in [LP09] [BHR12]: circuit-input wires (are on the receiving end of the circuit and process inputs x and y), circuit-outputs wires (that carry the circuit output value, denoted as $C(x, y)$), gate-inputs wires (that enter some gate g) and gate-output wires (that exit some gate g, carrying value $g(x, y)$).

Yao's protocol was initially designed as a two-party secure computations. In this setting, its participants are denoted as $garbler$ and $evaluator$. We will proceed now describing this protocol in high-level. The construction can be considered a $compiler$ that takes a polynomial function f in form of a circuit C that computes the respective f and constructs a protocol for securely computing f in the presence of semi-honest adversaries.(trebuie pus dupa ce se introduce notiunea). This protocol should ensure $privacy$ between participants. Therefore, all values v_i that are not allocated to circuit-output wires should not be determined by other parties. The main focus of Yao's protocol is to provide a way of computing any circuit C so that the values obtained on all wires other than circuit-ouput are never exposed. For every wire in the circuit, the $garbler$ chooses two random values such that one value substitutes the 0 and the other substitutes the 1. As an example, let us consider a wire labeled w. The $garbler$ chooses two values k_w^0 and k_w^1, where k_w^b represents the bit b. An essential observation is that even if one party knows the value k_w^b obtained by wire w, it is of no help in finding out if b is either equal to 0 or 1 because k_w^0 and k_w^1 are uniformly distributed at random. Now, let g be a gate with incoming wires w_1 and w_2 and with w_3 as its output wire. Given two random values ${k_{w_1}}^\alpha$ and ${k_{w_2}}^\beta$, it seems impossible for the $evaluator$ to compute such gate (and by extension, the whole circuit) because both α and β are unknown. Right away, the problem of computing the value of the output wire of a gate given the values of the two input wires to that gate arises. This is the part that involves $garbling$ (done by none other than the $garbler$ of the protocol) the values by providing a $garbled\ computation\ table$ that maps the random input values to random output values. However, such mapping shall have the property that given two input values only the output value that corresponds to the output of the respective gate is learned, keeping the other output value secret. To do this, the four possible input values to the gate g ${k_{w_1}}^0$, ${k_{w_1}}^1$, ${k_{w_2}}^0$, ${k_{w_2}}^1$ are viewed as encryption keys. Then, the outputs $k_{w_3}^0$ and $k_{w_3}^1$ (that are also keys) are encrypted with the respective keys from the incoming wires.

The following example is a garbled computation table for an OR gate g:

input wire w_1	input wire w_2	output wire w_3	garbled computation map
${k_{w_1}}^0$	${k_{w_2}}^0$	${k_{w_3}}^0$	$E_{{k_{w_1}}^0}(E_{{k_{w_2}}^0}({k_{w_3}}^0))$
${k_{w_1}}^0$	${k_{w_2}}^1$	${k_{w_3}}^1$	$E_{{k_{w_1}}^0}(E_{{k_{w_2}}^1}({k_{w_3}}^1))$
${k_{w_1}}^1$	${k_{w_2}}^0$	${k_{w_3}}^1$	$E_{{k_{w_1}}^1}(E_{{k_{w_2}}^0}({k_{w_3}}^1))$
${k_{w_1}}^1$	${k_{w_2}}^1$	${k_{w_3}}^1$	$E_{{k_{w_1}}^1}(E_{{k_{w_2}}^1}({k_{w_3}}^1))$

Table 2.1: Garbling an OR Gate

Key ${k_{w_3}}^1$ is encrypted under the pairs of keys associated with the values (0,1), (1,0) and (1,1) while the key ${k_{w_3}}^0$ is encrypted only under the pair of keys associated with (0,0).

Looking at table 2.1 we have a few observations. Firstly, given two input wire keys $k_{w_1}^{\alpha}$ and $k_{w_2}^{\beta}$ corresponding to α and β and the four values of the last column of table 2.1 we can decrypt and learn the output wire key $k_{w_3}^{g(\alpha,\beta)}$. And, as stated before, this is the only value that can be obtained because the other keys on the input wires are not currently known (so only one table value can be decrypted). Strictly speaking, it is possible for the $evaluator$ to compute the output key $k_{w_3}^{g(\alpha,\beta)}$ of a gate without learning anything new about the real values of α, β or $g(\alpha,\beta)$. Also, the order of the rows in such table can be randomly shuffled, so that it does not denote a particular pattern of placing keys.

Having described how a single garbled gate is constructed, let us proceed with illustrating how a garbled circuit is constructed. A garbled circuit consists of multiple garbled gates along with a set of *output decryption tables*. These are tables that map the random values obtained on the *circuit-output wires* back to their corresponding real values. So, for a $circuit\text{-}output$ wire w, the pairs (0,k_w^0) and (1,k_w^1) are granted by the $garbler$. After the $evaluator$ obtains the key k_w^{λ} on a circuit-output wire, he can determine the real value of the bit λ by looking at the output decryption table. Note that given the keys for the inputs x and y, it is possible to compute the entire circuit C going through every gate. Then, decrypting the keys obtained on the circuit-output wires, he can determine the value of the circuit, $C(x,y)$.

A more formal way of looking at this protocol is that of considering it as a set operations on "locked boxes". As stated above, the basic idea is to consider that every wire has two padlock keys associated with it: one key represents the 0 bit while the other represents the 1 bit. Then, four doubly-locked boxes are provided for each gate present in the circuit, where each box represents a row of the truth table computing gate (i.e., one box is related with input (0,0), another with (0,1) and so forth). The four boxes are locked in such way that each pair of keys (representing the inputs of the wires) opens exactly one box. Then, in each box there is placed a single key associated with the output wire of the respective gate. The key is chosen so that it represents the correct logical output given input the two keys (given two keys associated with 1 and 0 that open a box and a gate that computes the OR function, the "result" key inside the box is associated with 0 in the output wire). A notable observation is that, in this case, $computing\ the\ circuit$ means opening the locked boxes one at the time using the set of keys associated with parties' inputs (only one box opens for each gate). This process concludes with opening the last box, the one associated with the output gate, that can contain the actual output rather than another key. A second important observation is that in this process of computing the circuits no information is revealed but the output itself. This is due to the fact that no key is labeled, essentially making it impossible to learn if a certain key is associated with zero or with one and to reveal any association with the parties' inputs. Also, assuming only one set of keys is provided, in each gate only one box can be opened. In the actual circuit construction, the double-locked boxes represent the double-encryption and the padlock keys serve as decryption keys.

We will now proceed to informally describe Yao's protocol. As we noted earlier, one party is called $garbler$ and has the role of constructing the circuit and sending it to the other party, the $evaluator$. Then, the two of them interact so that the $evaluator$ obtains the input-wire keys that are associated with inputs x and y. With these keys, the $evaluator$ computes the circuit as depicted and obtains the output, finishing the protocol. At this point, a way of communicating the output of the $garbler$ must be defined, since we only focused on how the $evaluator$ receives his output. The $evaluator$'s output can contain the $garbler$'s output in an $encrypted\ form$ (such that only the $garbler$ knows the decryption key). Then, the $evaluator$ can just send the whole output back to the $garbler$ at the end of the protocol. Since it is encrypted, the $evaluator$ learns nothing more than its own output, which is what we intended.

The process of receiving keys associated the circuit-input wires by the $evaluator$ can cause several problems as noted in [Kur14] [LP09]. Firstly, we make a distinction between inputs of the $evaluator$ and the inputs of the $garbler$. In the $garbler$'s case, it will send the $evaluator$ the values that correspond to its input. That necessarily means that if the i^{th} bit of the input is 0 and the wire that receives this input is w_i, the $garbler$ provides the $evaluator$ with the string k_i^0. Since that all the keys follow a uniform random distribution, the $evaluator$ is able to learn nothing about the $garbler$'s input from these keys. Now, a security problem arises in the case of the $evaluator$. The $garbler$ cannot simply send all its keys related to its input (ex: both 0 and 1 keys on the $evaluator$'s input wires) because that would allow the $evaluator$ to compute more than its input, which in turn reveals more information than allowed. For example, for a given x, input of the $garbler$, receiving the keys would enable the $evaluator$ to compute $C(x, \overline{y})$ for every $\overline{y}$. That means learning much more information than a single value $C(x, y)$, violating the $privacy$ property. On the other hand, the $evaluator$ cannot just ask the $garbler$ for the appropriate keys because he would learn the $evaluator$'s input. This problem can be addressed using a 1-out-of-2-oblivious transfer protocol 1.1. In this protocol, a sender inputs two values x_1 and x_2 (in our case, the keys k_w^0 and k_w^1 for some circuit-input wire w) and the receiver inputs a bit b (in our case, the appropriate input bit b). The objective of the protocol is that the receiver obtains the value x_b (in our case, k_w^b). Additionally, the receiver learns nothing about the other value, x_{1-b}, while the sender learns nothing about the receiver's input b. By having the $evaluator$ access the keys in a similar way, we obtain the privacy of inputs on the $garbler$'s side while the $evaluator$ only obtains a single set of keys which can help him compute the circuit on a single value, as required.

2.5.2 Secure Two-Party Protocols for Semi-Honest Adversaries

The definitions mentioned here are according to [Gol04]. [LP09]

Definition 2.5.1. *Two-party computation* [LP09]

A two-party computation protocol is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such process as a **functionality** and denote it: f : {0, 1}* × {0, 1}* → {0, 1}* × {0, 1}*, where $f = (f_1, f_2)$. That is, for every pair of inputs $x, y \in \{0, 1\}^n$, the output-pair is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x, y)$ and the second party (with input y) wishes to obtain $f_2(x, y)$. We often denote such functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, for the oblivious transfer functionality is specified by $((x_0, x_1), b) \mapsto (\lambda, b)$, where λ denotes the empty string. When the functionality f is probabilistic, we use the notation $f(x, y, r)$ where r is a uniformly chosen random tape used for computing f.

Definition 2.5.2. *Privacy by Simulation* [LP09]

Intuitively, a protocol is secure if whatever can be evaluated by a party participating in the protocol can be computed based on its input and output only. This is formalized according to the simulation paradigm. Loosely speaking, we require that a party's $view$ in a protocol execution be simulable given only its input and output, meaning that the ideal-world protocol (where there is a trusted party that receives all private inputs, including his) delivers the same results (returning private outputs to each party) as the real-world protocol. This then implies that the parties learn nothing from the protocol $execution$ itself, as desired.

Definition 2.5.3. *Definition of Security* [LP09]

Given the following notations:

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality, and let π be a two-party protocol for computing f.
- The view of the i^{th} party ($i \in \{1, 2\}$) during the execution of π on inputs (x, y) is denoted as view_i^π and equals $(x, r^i, m_1^i, m_2^i, ..., m_i^t)$, where r^i represents the contents of the i^{th} party's $internal$ random tape, and m_i^j represents the j^{th} message that party P_i has received.
- The output of the i^{th} party during execution of π on (x, y) is noted output_i^π and can be computed from its own view of the execution. Denote output_i^π = (output_1^π,output_2^π)).

Definition 2.5.4. *Security for semi-honest behavior*[LP09]

Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f in presence of semi-honest adversaries if there exists probabilistic polynomial time algorithms S_1 and S_2 such that:

$$\{S_1(x, f_1(x, y)), f(x, y)\}_{x,y\in\{0,1\}*} \stackrel{c}{\equiv} \{(\mathsf{view}_1^\pi(x, y))\}_{x,y\in\{0,1\}*} \tag{2.1}$$

$$\{S_1(x, f_2(x, y)), f(x, y)\}_{x,y\in\{0,1\}*} \stackrel{c}{\equiv} \{(\mathsf{view}_2^\pi(x, y))\}_{x,y\in\{0,1\}*} \tag{2.2}$$

where $|x| = |y|$ and $\stackrel{c}{\equiv}$ denotes *computational indistinguishability*.

Equations (2.1) and (2.2) state that the view of a party can be simulated by a probabilistic polynomial-time algorithm given access to the party's input and output $only$. We emphasize that the adversary here is semi-honest and therefore the view is exactly according to the protocol definition.

Deterministic same-output functionalities. A functionality $f = (f_1, f_2)$ is same-output if $f_1 = f_2$. To see this, note that given a protocol for securely computing a deterministic functionality, it is possible to construct a secure protocol for computing any probabilistic functionality.

2.5.3 The garbled circuit construction

In this part, we describe the garbled circuit construction as in [LP09]. Let C be a boolean circuit which receives two inputs x,$y \in \{0,1\}^n$ and outputs $C(x, y) \in \{0,1\}^n$(for simplicity, we assume that the input length, output length, and the security parameter are all of the same length n). Assuming that C has the property that if a circuit-output wire comes from a gate g, then gate g has no wires that are input to other gates. (Likewise, if a circuit-input wire is itself also a circuit-output, then it is not input into any gate.)

Since C is a boolean circuit, any gate contained by it is represented by a function $g : \{0,1\} \times \{0,1\} \to \{0,1\}$. Let w_1 and w_2 two input wires of gate g and the output of it w_3. Moreover, let $k_1^0, k_1^1, k_2^0, k_2^1, k_3^0, k_3^1$ be the six keys obtained by independently a key-generator algorithm G (for simplicity, assume that the lenght of each key is also n). The goal is to be able to compute $k_3^{g(\alpha,\beta)}$ from k_1^α and k_2^β, without revealing the other values, $k_3^{g(1-\alpha,\beta)}$,$k_3^{g(\alpha,1-\beta)}$,$k_3^{g(1-\alpha,1-\beta}$).

The gate g is defined by:

$$c_{0,0} = E_{k_1^0}(E_{k_2^0}(k_3^{g(0,0)})) \tag{2.3}$$

$$c_{0,1} = E_{k_1^0}(E_{k_2^1}(k_3^{g(0,1)})) \tag{2.4}$$

$$c_{1,0} = E_{k_1^1}(E_{k_2^0}(k_3^{g(1,0)})) \tag{2.5}$$

$$c_{1,1} = E_{k_1^1}(E_{k_2^1}(k_3^{g(1,1)})) \tag{2.6}$$

where E represents a private-key encryption scheme (G, E, D) that has indistinguishable encryptions under plaintext attacks. The gate is determined by randomly permuting the above set of values, denoted as c_0, c_1, c_2, c_3. These also determine the garbled table of the gate g. So, given values k_1^α and k_2^β and c_0, c_1, c_2, c_3, the output of the gate g, $k_3^{g(\alpha,\beta)}$, can be computed as follows: for every $i \in \{0, 1, 2, 3\}$, compute $D_{k_2^\beta}(D_{k_1^\alpha}(c_i))$. If one or more decryption fails (i.e. the value obtained cannot be given as an input for the next decryption), then the output will be $abort$. Otherwise, define k_3^θ as being $k_3^{g(\alpha,\beta)}$ (the correct output) and return it. We can now describe how to construct the entire garbled circuit. Let m denote the number $wires$ in the circuit C, with $w_1, w_2, ..., w_n$ being the labels of these wires. These labels must be chosen uniquely so that if any w_i and w_j are both output wires of a gate g, then that means $w_i = w_j$. On the same line, if an input bit enters more than one gate, then all circuit-input wires that are associated to that input bit shall have the same label. Then, for every wire label w_i there are chosen two keys k_i^0, k_i^1 by the generation algorithm $G(1^n)$, which are independent (in relation with the other generated keys). Next, given these keys, the garbled values of each gate are computed as above and the result is permuted at random. Now, the output decryption tables of the garbled circuit can be computed. These tables are populated with the values $(0, k_i^0)$ and $(1, k_i^1)$, where w_i is the label of a circuit-output wire.

The entire garbled circuit C, denoted $G(C)$, is represented by the garbled tables for each gate and the output tables. Notice that the structure of C is given and the garbled version of C is a map that specifies which table corresponds to each gate, and the output tables. Now, the description of the garbled circuit is finally complete.

2.5.4 Yao's Two-Party Protocol

As we observed, it is possible to obtain the correct output from the garbled circuit given the keys that correspond to the correct input. Thus, the protocol proceeds by party denoted by $garbler$ constructing the garbled circuit and giving it to the other party, denoted by $evaluator$. After that, the $garbler$ hands $evaluator$ the keys that correspond to input $x = x_1, x_2...x_n$. Also, the $evaluator$ has to obtain the keys that correspond to its input $y = y_1, y_2, ..., y_n$ while ensuring the following, as pointed in [LP09]:

- The $garbler$ should not learn anything new about the $evaluato$'s input, y.

- The *evaluator* should obtain the keys corresponding to y and no others (otherwise, he could compute $C(x, y)$ and $C(x, y')$ with $y \neq y'$).

The above two problems are solved by having both the *evaluator* and the *garbler* run a *1-out-of-2 Oblivious Transfer* (1.4). That is, for every bit of the *evaluator*'s input, the parties run an oblivious transfer where the *garbler*'s input is (k_{n+i}^0, k_{n+i}^1) and the *evaluator*'s inputs is y_i. This way, the *evaluator* obtains the keys $k_{n+1}^{y_1}, ..., k_{2n}^{y_n}$ and only these keys. In addition, the *garbler* learns nothing about y.

Protocol 4 Yao's two-party protocol

Inputs. The garbler has $x \in \{0,1\}^n$ and the evaluator has $y \in \{0,1\}^n$.

Auxiliary input. A boolean circuit C such that every for every $x, y \in \{0,1\}^n$, it holds that $C(x, y) = f(x, y)$, where $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. C is required to be such that if a circuit-output wire leaves some gate g, then gate g has no other wires leading from it into other gates (no circuit-output wire is also a gate-output wire). Furthermore, a circuit-input wire that is also a circuit-output wire enters no gates.

The protocol:

1. The garbler constructs the garbled circuit $G(C)$ as described in section 2.5.3 and sends it to the evalutator.

2. Let $w_1, w_2, ..., w_n$ be the circuit-input wires corresponding to x and $w_{n+1}, ..., w_{2n}$ be the circuit-input wires corresponding to y. Then, the garbler sends to evaluator the strings $k_1^{x_1}, ..., k_n^{x_n}$.

3. For every i, both parties execute a *1-out-of-2 oblivious transfer* in which the garbler's input equals (k_{n+i}^0, k_{n+i}^1) and the evaluator's input is y_i.

   (The above oblivious transfers can be run in parallel.)

4. Now, the evaluator obtains the garbled circuit and the $2n$ keys corresponding to the $2n$ input wires of C. The evaluator computes the circuit as described in 2.5.3 obtaining $f(x, y)$.

5. The evaluator sends the output f to the garbler.

Theorem 1. *[LP09] Let f be a deterministic same-output functionality. Furthermore, assume that oblivious transfer protocol is secure in the presence of semi-honest adversaries and that the encryption scheme has indistinguishable encryptions under chosen plaintext attacks and has an elusive and efficiently verifiable range. Then, Yao's two party protocol (2.5.4) securely computes f in presence of semi-honest adversaries.*

2.6 The multiparty case

2.6.1 What if parties do not follow instructions?

Up until now we assumed that parties would be benevolent and follow what they are instructed to do. But, as said in [CDN15], this is not a reasonable assumption because it can lead to dangerous situations, where one party may have an interest in doing something different than what they are told to do. We will consider two different particular scenarios:

Choosing inputs

Considering the matchmaking protocol presented in section 2.4 and assuming that two parties, P_1, P_2, are participating in the protocol but P_1 is not really interested in collaboration with P_2, while P_2 is more open in that sense, it is easy to observe that party P_1 can choose an input that does not represent its actual inputs, being $dishonest$ while doing so: P_1 could choose $a = 1$ as its input, in which case ab mod $p = b$, so that now it learns b, the P_2's private input.

It is interesting to note that, on second thought, it is impossible to distinguish honest and valid input from dishonest and malevolent input. Whatever protocol we design, a party can always choose its input on its own, with no regulation whatsoever. Therefore, party P_2 wants to participate in a *2-party* protocol and wants its input to be safe, then it should not $participate$ at all!

Moreover, to do secure computation, we have to assume that parties will try to obtain $meaningful$ output, meaning that they provide inputs in a $reasonable$ way, such that it makes sense with regards to their goal.

Deviation from the protocol

No matter how parties choose their inputs, some problems may arise when a party tries to deviate from the protocol to obtain more information about others' inputs, or force the computation to give a wrong result, as noted in [CDN15]. Such problems are ample, but, in contrast to the issues of dishonest input choices, we have several options to handle them.

A great way to solve these issues is to add mechanisms to the protocol that ensure that any deviation from the protocol is detected and reported. To exemplify this, we will tackle a case presented in [CDN15] for Protocol Secure Addition. In this protocol, we first ask parties to distribute $shares$ of their respective inputs to other parties present. Looking at P_1, he picks shares $r_{1_1}, r_{1_2}, r_{1_3}$ such that his private input x_1 is equal to $r_{1_1} + r_{1_2} + r_{1_3}$ mod p; then, he must send r_{1_1}, r_{1_3} to party P_2 and send r_{1_1}, r_{1_2} to party P_3. Firstly, P_1 could choose shares $r'_{1_1}, r'_{1_2}, r'_{1_3}$ such that $x_1 \neq r'_{1_1} + r'_{1_2} + r'_{1_3}$ mod p. This is not really a problem, since it means he used another value, x'_1, as its private input with $x'_1 = r'_{1_1} + r'_{1_2} + r'_{1_3}$ mod p. As stated above, there should be no restriction on which input does P_1 choose. The second way in which P_2 can deviate from the protocol is to send r_{1_1}, r_{1_3} to P_2 and send r'_{1_1}, r'_{1_2} to P_3, with $r_{1_1} \neq r'_{1_1}$. Since the input x_1 is not well defined, this problem is more severe. This might or might not lead to an attack but it is at least a case of clear deviation from the protocol. However, we can catch such party by making the following verification: when P_2 receives share r_{1_1} from P_1, he sends it to P_3 and requests the share r_{1_1} that he received, for verification purposes. Then, they can check if they hold the same value. Similarly, each pair of party can check for consistent shares. In general, having parties communicate more with each other can make a protocol more vulnerable to insecurities, but this is not the case because each party sends information that the other should already have had.

After this phase, parties are asked to add their shares and make the resulted sums public. Other deviations can occur at this point. P_1 is asked to evaluate s_2 and s_3 and make the public, but he might deviate and broadcast s'_2 instead of s_2 with $s'_2 \neq s_2$. This could lead to a wrong result for all other parties involved, $s = s_1 + s'_2 + s_3$ mod p. Note that P_3 also has to evaluate p_1 and P_2 and make them public. So, parties can check if s_2 is the same for both parties. Similarly, all parties can check if s_1 and s_3 are evaluated correctly (the two versions broadcasted are identical).

The above means that Protocol secure addition has the following property: if any single party deviates from the protocol and does what it is not supposed to do, the other parties will be able to detect this. This idea of pinging other parties to ensure that the protocol is followed is an occurring theme in the fields of secure multiparty computation.

We have only addressed what a *single* party can do to harm the computational process. On the following pages, there will be presented some results that address a more general case where a certain number of parties try, for instance, to compute information on other parties' inputs.

2.6.2 Adversarial Power

The aim of *secure multiparty computation* is to enable parties to compute distributed compute task in a secure manner. But, as noted in [Lin09], when distributed computed deals with concerns of computing tasks under the threat of hardware crashes and other failings, secure multiparty computation protocols deal with the possibility of deliberately malicious behaviour by some adversarial party. That means, it is assumed that such protocol may come under *attack* by an external entity, or even a subset of present parties. The main purpose of such attack is learn private information about other parties' inputs or lead to make the result of the computation incorrect. Hence, as already said in section 2.3, the two important requirements of any secure multiparty computation are *privacy* and *correctness*. The *correctness* property requires means that each party shall receive the correct output. Therefore, no adversary should be able to disrupt the computation process such way that it deviates from the function that parties had agreed on evaluating. The privacy property says that nothing should be learned but what is absolutely necessary.

The context of secure multiparty computation covers tasks as complex as secure voting (2.3), electronic auctions (2.4), anonymous transactions and private information retrieval schemes (1.4). Let us consider the tasks of voting and auctions. The *privacy* requirement for an election protocol establishes that no party learns anything about the original votes of other parties, while the *correctness* property establishes that no set of parties can control the outcome of the election (beyond, of course, voting their favored candidate). Additionally, in the auction scheme, the privacy property ensures that the winning bid is revealed, while the *correctness* property ensures that the party that wins the auction is the one with the highest bid. Due to its general scope, the secure multiparty model can denote almost any cryptographic problem.

Security in multiparty computation model

As presented in [Lin09], the model that we consider is the one where some adversarial entity is controlling a small portion of the involved parties in the protocol and wants to do harm to the protocol execution. Parties under such control are called corrupted, and always follow the adversaries' instructions. Protocols should resist any attack. In order to formally define a protocol as being $secure$, the definitions in section 2.3, namely $privacy$ and $correctness$, are not enough. We now must add some more properties:

- *Independence of inputs*: Corrupted parties must choose their inputs independently of the private parties' inputs.
- *Guaranteed output delivery*: Corrupted parties should not be able to prevent honest parties from receive their input. Specifically, the adversary shall not be able to disrupt the computation by carrying out a *denial-of-service* type attack.
- *Fairness*: Corrupted parties should receive their outputs if and only if the honest parties receive their outputs. That way, a situation where a honest party does not receive an output while a corrupted one does should not be allowed to happen. This property in essential in many protocols.

The above requirements do not represent a definition of security. But, they should hold for any secure protocol. Moreover, it is easy to see that there is no finite list of requirements to be fulfilled for any secure protocol, and we should not make one.

Adversarial power.

The above definition of security disregards one important problem: the power of the adversary that attacks a protocol execution. We now describe the corruption strategy (or how the parties get under his control), the allowed adversarial behaviour and what the complexity of such adversary is assumed to be (i.e., polynomial-time or computationally unbounded):

- **Corruption strategy**: The corruption strategy approaches questions such as when or how are parties corrupted. Two main models are:
  - Static corruption model: In this case, the adversary is given a set of finite number of parties that he can control during the execution of the protocol. Parties cannot change states (honest parties remain honest while corrupted parties remain corrupted).
  - Adaptive corruption model: The choice of who to corrupt and when can be decided on the spot by the adversary, depending on its view of the execu-

tion. Note that in this model, once a party has been corrupted, we consider it to remain corrupted through the rest of the protocols' execution.

  – Active corruption model[Bar+14]: In this case, a party can be considered corrupted in an interval of time only; therefore, honest parties can become corrupted through the computation, but later they might be considered honest again.

- **Allowed adversarial behaviour**: A parameter that defines the actions of corrupted parties; there are two main types of adversaries:

  – Semi-honest adversaries: In this model, even parties that are considered corrupted do follow the protocol specification. However, the adversary obtains the internal state of all corrupted parties and attempts to use this information to learn new private information. This a weak adversarial model, but there are some settings in which it can pose a threat for some protocols. Semi-honest adversaries are often denoted as *passive* or *honest-but-curious*.

  – Malicious adversaries: In this model, the corrupted parties can deviate from the protocol's specification whenever the adversary instructs them to do so. Essentially, it is desired to provide protection in presence such adversarial power because it ensures that no attack can succeed. Malicious adversaries are also called *active*.

- **Complexity**: Lastly, the two categories of assumed adversarial computational complexity are:

  – Polynomial-time: The adversary is allowed to run in polynomial-time.

  – Computationally unbounded: The adversary has no computational limits.

Chapter 3

Application

3.1 Data mining

Various amounts of data is collected in hopes of presenting a benefit for many organizations (i.e. discovering new hidden patterns of said data). Data mining as a whole has proven to be a great advancement in domains such as big data, internet of things and security. But, with the rise of regulations of the use of private customer data such as GDPR, mining on private data can become a tricky problem. Data mining as a whole very rarely violates the privacy of independent data since it recovers the underlying patters of such data. The main focus of the following application is to introduce a protocol based on cryptography and statistics to enable secure clustering on the private data of individuals.

3.1.1 KMeans algorithm

KMeans is a classic example of a clustering algorithm. In this model, the goal is to partition the data points into k clusters (sets of data points). Each cluster is associated with a mean (often denoted centroid). We will proceed with describing the algorithm, as in [JD88]:

Protocol 5 KMeans

Inputs. k, the number of desired clusters, n the number of data points and the data points.

Goal. Obtain k clusters alongside k means.

The protocol:

1. Initialize the k means $\mu_1 ... \mu_k$ to 0.
2. Arbitrarily select k starting points $\mu'_1 ... \mu'_k$
3. repeat
   - Assign $\mu'_1 ... \mu'_k$ to $\mu_1 ... \mu_k$ respectively
   - for all *points* i
     - put point i in the *closest cluster* (given a distance function)
   - end for
   - Calculate new means $\mu_1 ... \mu_k$

until the difference between $\mu'_1 ... \mu'_k$ and $\mu_1 ... \mu_k <$ an arbitrary threshold

k-means can be seen as an optimization algorithm since its goal is to minimize the sum of squared distances of each data point to its nearest mean.

3.1.2 Privacy-preserving clustering over vertically partitioned data

The difference between vertically and horizontally partitioned data is that while on the horizontally partitioned model the parties have the same database scheme with different entries, the vertically partitioned model says that the parties have entries that refer to the same entities but each party has different attributes. For example, consider a number r of clinics that have databases that contain entries for the same set of n customers. While some clinics can be specialized on different types of psychological consultations that give different diagnostics (such as predisposition to depression, autism and anxiety), others can have a thorough analysis over a patients' physical and cardiovascular conditions (their databases could contains attributes such as heart, renal, etc. failure risk, asthma, diabetes, etc.). What those parties would be interested in

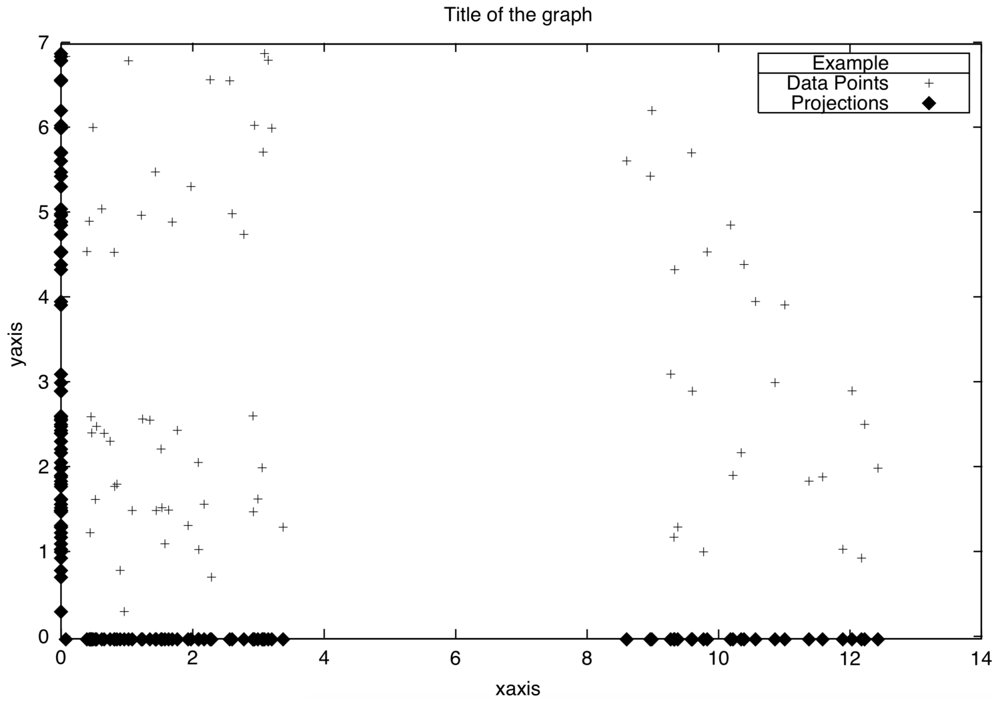


Figure 3.1: From [VC03]

learning is a *clustering* over their distributed data, without violating the privacy property of the individual data of each affected customer. Sure, one may say that, since their databases' attributes differ, each party could run a clustering algorithm to locally determine the clusters' composition. But, as seen in figure 3.1, this is not the same as running the same algorithm over the centralized distributed data. Looking at the y's axis, there appear to be two major clusters. But, as a whole, it is easy to observe that y interpretation of said clusters' composition is wrong; when aggregating all attributes, the true clusters are different.

We will now begin to formally describe our problem as in [VC03]. Given r parties with different attributes for the same number of entities. Let n be the number of entities and k the number of clusters. The participating parties want to cluster their *distributed* data using the k-means algorithm. The final and desired result should consist in the assignment of entities to clusters and, locally, the position of each mean/centroid associated with each cluster. Let each cluster mean be μ_i, with $i = 1, ..., k$. Also, μ_{ij} represents the value of mean of the cluster i on party j. Notice that the composition of the clusters should be the same for all parties, but their means differ because we are talking about vertically partitioned data. So, the final result for a party j is:

- the final position of μ_{ij}, $i = 1, ..., k$

- cluster assignments: *clust*$_i$ for all data points $i = (1, ..., n)$

Initially, the algorithm requires an assignment for the positions of the k means. This is an important decision, since, as noted in [BF98], the initial assignment can lead to a different final configuration of clusters and a different set of final means. Many techniques have been proposed for solving this problem [YPS10], but, for simplicity, we will assume that the initial k means are selected arbitrarily. So, every party j chooses every mean μ'_{ij} as he wishes, with $i = 1, ..., k$. These values are chosen independently of others and are therefore private.

The algorithm 1 follows the standard k-means algorithm. We present its secure variation, as described in [VC03]. The means are updated at each iteration, with the objective of minimizing the sum of the squared distances between the current means and the true means. To do this, at each iteration every data point is placed in the *closest cluster*, in relation with the distance function. Once the cluster composition is known, the means can be updated locally by each party. For the termination criteria, we use the $checkThreshold$ to securely check if the last iteration of the algorithm produced an improvement to the mean approximation below a certain threshold.

Algorithm 1: Privacy-preserving k-means clustering

Input: r parties, n data points, k clusters

Result: k means and the cluster's composition

```
for all parties j = 1, ..., r do
    for all clusters i = 1, .., k do
        initialize μ'_ij
    end
end
repeat
    for all parties j = 1, ..., r do
        for all clusters i = 1, .., k do
            μ_ij ← μ'_ij;
            Cluster[i] ← ∅;
        end
    end
    for all points g = 1, .., n do
        for all parties j = 1, ..., r do
            {Compute the distance vector X⃗_j from point g to each cluster};
            for all clusters i = 1, ..., k do
                x_ij = dist(g, μ_ij);
            end
        end
        Each party puts g into Cluster[closestCluster] {algorithm 3};
    end
    for all parties j = 1, .., r do
        for all clusters i = 1, ..., j do
            μ'_ij ←mean of j's attributes for points inCluster[i] ;
        end
    end
until checkThreshold is True {algorithm 2};
```

Termination

The $checkThreshold$ algorithm 2 represent the classic distributed approach: centralize

Algorithm 2: checkThreshold: Are the new means sufficiently close to old means?

Input: r parties, n data points, k clusters, Th threshold for termination, $rand$ random number generator that produces values uniformly distributed over $0..n-1$ over the domain of the distance function D

Result: True if the threshold is smaller than the global error, false otherwise

```
for all parties j = 1, ..., r do
    err_i ← 0;
    for all clusters i = 1, .., k do
        err_i ← err_i |μ'_ij −_D μ_ij|;
    end
end
{Securely compare ∑ err_j < Th.};
At P_1 : m = rand(); for all parties j = 2,...,r-1 do
    P_i sends m + err_j( mod n) to P_{j+1};
end
At P_r: m = m + err_r;
P_r broadcasts Th < m to all other parties.
```

all errors and compare with the threshold Th. To maintain security, all operations are done mod n, where n, in this setting, represents the maximum value of the domain of the distance function D.

Securely Finding the Closest Cluster

This function of the k-means algorithm is called for every data point to privately find the closest cluster. Each party has as private input a vector containing the distances from this point to every means.

The problem is formally defined as presented in [VC03]: Considering r parties $P_1, P_2, ..., P_r$, each with their own vector $\vec{X}_i$ for any data point g:

$$P_1 \text{ has } X_1 = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{k1} \end{bmatrix}, P_2 \text{ has } X_2 = \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{k2} \end{bmatrix} \; ... \; P_r \text{ has } X_r = \begin{bmatrix} x_{1r} \\ x_{2r} \\ \vdots \\ x_{kr} \end{bmatrix}$$

The goal is to compute the index l of the row with the minimum sum:

$$\underset{i=1..k}{\operatorname{argmin}}\Big(\sum_{j=1..r} x_{ij}\Big)$$

The security algorithm has three main ideas:

- Disguise the values of the distance vector $\vec{X}_i$ for each party i with random values that cancel out when combined.
- Compare the distances to find the minimum in a secure manner using garbled circuits.
- Permute the order of clusters so the real meaning of the comparison is unknown until the end.

This part of the algorithm requires three non-colluding parties. We will consider that P_1, P_2 and P_r to be non-colluding and semi-honest. We continue with the steps of the algorithm as presented in [VC03]. Party P_1 generates a length k vector $\vec{V}_i$ for each party i, such that $\sum_{i=1}^{r} \vec{V}_i = \vec{0}$. P_1 also generates a permutation π over the elements {1,...,k}. Then, P_1 engages each party using the algorithm 5 to generate the sum of $\vec{V}_i$ and $\vec{X}_i$. The resulted vector, $\vec{T}_i$, is only known to P_i but it is permuted by π only known to P_1. Parties P_1,P_3 to P_{r-1} send their vectors to P_r

Parties P_2 from P_r engage in a series of secure addition and comparisons to find the index of the row with the minimum sum of distances. Formally, what they would like to know is wheather or not $\sum_{i=1}^{r} x_{li} + v_{li} < \sum_{i=1}^{r} x_{mi} + v_{mi}$. Since , $\sum_{i=1}^{r} v_{li} = 0$, the real result is $\sum_{i=1}^{r} x_{li} < \sum_{i=1}^{r} x_{mi}$, showing which cluster, l or m, is closer to the point g.

Party P_r has now all the components of the sum except $\vec{X}_2 + \vec{V}_2$. For each comparison, we need a secure circuit evaluation that calculates operations of the following form: $a_2 + b_r < b_2 + a_r$. After $k-1$ of such secure comparisons, the minimum cluster is known by P_2 and P_r. But, what they currently know is the permuted minimum. So, P_r (or P_2) needs to send the minimum to P_1. P_1 now broadcasts the real minimum, $\pi^{-1}(minimum)$ to all parties and each party puts point g into $Cluster[minimum]$.

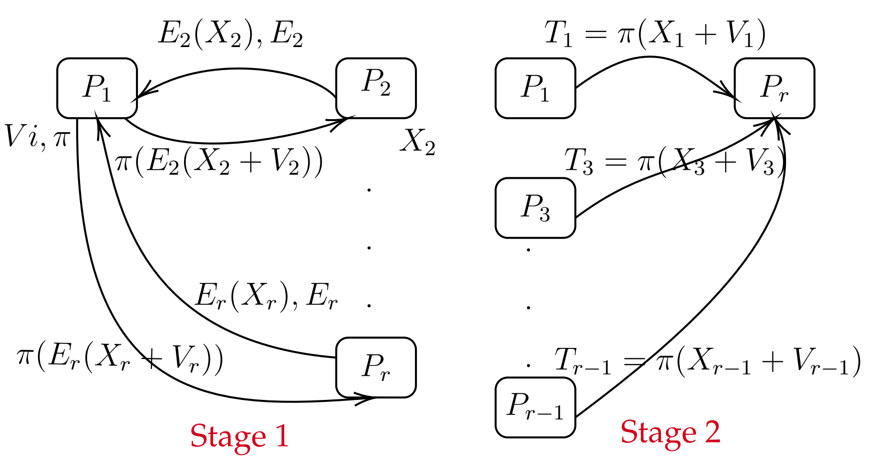
Algorithm 3: closestCluster: Find minimum distance cluster

Input: r parties,each has a length k vector $\vec{X}$ of distances to point g, k clusters.

Three of these parties are trusted not to collude: P_1, P_2 and P_r

Result: the minimum distance cluster index

{Stage 1: Between P_1 and all other parties};
P_1 generates a random permutation π over $1, ..., k$;
P_1 generates r random vectors $\vec{V_i}$ summing to 0 (algorithm 4);
for *all parties* $i = 2, ..., r$ **do**
  $\vec{T_i}$(atP_1) = $addAndPermute((\vec{V_i}, \pi$ (at P_1),$\vec{X_i}$(at P_i)) (algorithm 5) ;
end
{Stage 2: };
for *all parties i=1,3,...,r-1* **do**
  P_i send $\vec{T_i}$ to P_r
end
P_r computes $\vec{Y} = T_1 + \sum_{i=3}^{r} \vec{T_i}$;
{Stage 3:};
minimum $\leftarrow$ 1;
for *all clusters j= 2,...,k* **do**
  if *secureAddAndCompare*$(Y_j + T_{2j} < Y_{minimum} + T_{2minimum})$ **then**
    minimum $\leftarrow j$
  end
end
{Stage 4:};
P_r sends *minimum* to P_1;
P_1 broadcasts $\pi^{-1}(minimum)$ to all other parties;



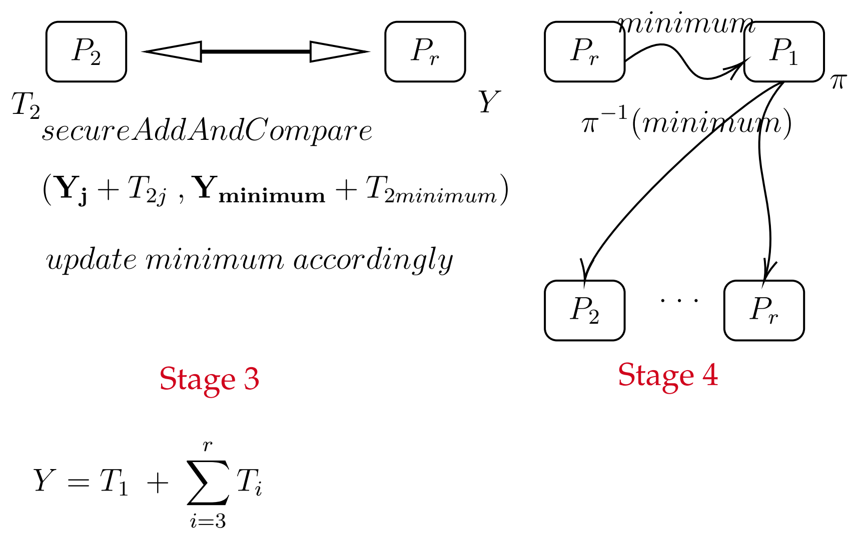
with $V_{k \times r}$ such that $\sum_{i=1}^{r} \vec{V_i} = \vec{0}$

Algorithm 4: genRandom: Generates a random matrix $V_{k \times r}$

Input: random number generator $rand$ that produces values uniformly distributed over $0..n-1$ spanning the domain of the distance function D

Result: $V_{k \times r}$ with the property that $\sum_{i=1}^{r} \vec{V_i} = \vec{0}$

```
for all i = 1, ..., k do
    PartSum_i ← 0 for all j = 2, .., r do
        _ij ← rand();
        PartSum_i ← PartSum_i + _ij mod n
    end
    V_i1 ← PartSum_i mod n
end
```



Permutation algorithm

The permutation algorithm used as described in [VC03] is an asymmetric two-party algorithm, defined as follows: there are two parties, A and B. B has an n-dimensional vector $\vec{X} = (x_1, ..., x_n)$, and A has an n-dimensional vector $\vec{V} = (v_1, ..., v_n)$. A also has a permutation π of the n indices. Their goal is to give party B $\pi(\vec{X} + \vec{V})$, without disclosing each others' private inputs (i.e. B should not learn π or $\vec{V}$, while A should not learn $\vec{X}$). For our algorithm, $\vec{V}$ represents a random vector from a uniform random distribution.

One possible way to solve this problem is to make use of the SMC primitive named Homomorphic encryption and described in section 1.5.

Algorithm 5: permutationAlgorithm

Input: two parties A and B with their inputs: π and $\vec{V}$, $\vec{X}$ respectively

Result: B learns $\pi(\vec{X} + \vec{V})$

1 B generates a public-private keypair (E, D) for a homomorphic encryption scheme.;

2 B encrypts its vector $\vec{X}$ and obtains $\vec{X}'$, with $\forall i, x_i' = E(x_i)$;

3 B sends $\vec{X}'$ to A alongside the public key E;

4 A encrypts its vector $\vec{V}$ and obtains $\vec{V}'$, with $\forall i, v_i' = E(v_i)$;

5 A multiplies the components of $\vec{X}'$ and $\vec{V}'$ to get $\vec{T}'$, with $\forall i, t_i' = x_i' + v_i'$. A applies the permutation π over $\vec{T}_i'$ to get $\vec{T}_p'$;

6 A sends $\vec{T}_p'$ to B;

7 B decrypts the vector element by element and obtains $\vec{T}_p'$, $\forall i, t_{pi} = x_{pi} + v_{pi}$;

At line 5 of the permutation algorithm , due to additively homomorphic property of the encryption scheme, the following property applies:

$$t_i' = x_i' + v_i' = E(x_i) + E(v_i) = E(x_i + v_i)$$

Definition 3.1.1. [VC03] The permutation algorithm 5 reveals nothing to party A.

Proof. *A's view*: A receives a public key E and an encrypted vector $\vec{X}$ of size n. □

Termination criterion

Definition 3.1.2. [VC03]

Algorithm checkThreshold determines if $\sum |\mu_{ij}' -_D \mu_{ij}| < Th$, revealing nothing except this result.

Proof. [VC03] Steps 11 to 14 are the only steps that require communication. In step 11, party P_1 first sends $m + d_j \pmod{n}$ where m is the random number known only to P_1. Each party P_i with $i \in \{2, ..., r-1\}$ send their message $m + \sum_{j=1}^{i} err_j \bmod n$. Since m is randomly chosen of the distribution of $0, ..., n-1$, this sum can be simulated by generating a random number of that distribution:

$$Pr[VIEW_j^{Algorithm2step11} = x] = Pr[m + \sum_{i=1}^{j} err_i = x] = Pr[m = x - \sum_{i=1}^{j} err_i] = 1/n$$

Also, the *secureCompare* function only gives the comparison output between $m(atP_r)$ and Th, as demonstrated in [LP09].

□

Algorithm 3 returns the index of the closest cluster given a given point g, while the distances are distributed across all parties r.

Definition 3.1.3. [VC03] Algorithm 3 is private and outputs the correct index of the cluster with the minimum sum of squared distances to each point in the data set.

Proof. [VC03]

- Stage 1: The only communication occurs in the calls of the Permutation Algorithm 3. What remains to show is that the resulting vectors $\vec{T_i}$ can be simulated. The simulation of P_1 remains P_1, while, for the remaining sites, we will also have a uniform distribution for $x_i + v_i$ over 0,...,n-1. So, for each party, vector $\vec{T_i}$ can be simulated by selecting random numbers over $0, ..., n-1$. This indistinguishable from what P_1 sees from the algorithm itself.

- Stage2: All parties other than P_2 and P_r send their information to P_r. So, the only concern at this stage is the view of the party P_r. His vector $\vec{Y}$ contains the sum of the actual distances mod n but T_2. Yet, $\vec{T_2}$ is also distributed randomly over 0,...,$n-1$, while $\vec{Y}$ represents $\vec{distances} - \vec{T_2}$, and, thus, it is also distributed randomly over 0,...,$n-1$. So, the view of P_r is indistinguishable from the real algorithm.

- Stage 3: Here, $P2$ and P_r engage on a series of comparisons. Each comparison is secure, as noted in section 2.5.3 and [LP09]. The simulator uniformly chooses a random ordering of k clusters from $k!$ possible orderings. The probability of any given order of clusters is $1/k!$, the same probability of any given ordering achieved by the permutation π. Therefore, the probability of any given sequence of comparisons results is the same under the view seen by the actual algorithm.

- Stage 4: P_r sends the index $minimum$ to P_1. Since the true index is the final result known to all parties, all parties can simulate it.

  We also have the problem of simulating the permutation π which is secure.

  Given these settings, algorithm 3 is privacy-preserving.

$\square$

Handling collusion

As noted in [VC03], parties P_1 and P_r have more information than others through the course of the protocol execution. P_1 knows

- the permutation π
- the random values of matrix $V_{k\times r}$

while P_r learns

- the permuted vectors of the permutation algorithm ($\vec{T_i}$) for all parties except P_2
- the comparison result

This information is meaningless in isolation. But, collusion between P_1 and P_r can be enough to derive the information regarding each party's distance to means from each point. Because of this, choosing the non-colluding parties is an important task.

A more technical solution is to choose a new party to play the role of P_1. Let p, $1 \leq p \leq r-1$. Thus, the permutation π and the matrix $V_{k\times r}$ is chosen differently for each iteration of the algorithm, while the sum of $\sum_{i=1}^{r}$ is still $\vec{0}$, so the total sum is still the actual distance. The final minimum index is $\pi_1^{-1}(\pi_2^{-1}(...(\pi_p^{-1}(i')...))$.

3.1.3 Implementation

The above k-means variation was implemented in Python. To simulate each participating party in the protocol, the communication between them is established using the zmQ python library ([Hin19]) that enables an easier approach to peer-to-peer communication using sockets. Each party has to know its corresponding $partyNumber$ and r, the total number of present parties, so that it can assume its role in the protocol; remember, P_1, P_r and P_2 have some extra computation responsibilities in this clustering protocol. Regarding the Stage 1 of the $closestCluster$ algorithm, we use the Pailier cryptosystem [Pai99b], which is additively homomorphic, through the *python-pailier* [19] library. The secure comparisons during Stage 3 are done using the [WMK16] library in order to remain true to the protocol definition and use a a garbled circuit for this operation.

We simulated a setting in which r medical clinics have information about the same set of patients and want to run k-means algorithm to obtain their relative means. Their database structure contains different attributes because the clinics are specialized in different types of analysis and consultations. Because of privacy reglementations such as GDPR, they do not want to expose data of individuals (knowing that they have a common set of patients does not violate this concern).

We used a public dataset from HealthData.gov. Its scheme contains various attributes such as: PatientID, Gender, Age, CCCount, RiskCount, ERCount, CCArthritis, CCAsthma, CCAtrialFibrillation, CCAutism, CCCancer, CCCOPD, CCDementia, CCDepression, CCDiabetes, CCHeartFailure, CCKidneyDisease, CCOsteoporosis, CCSchizophrenia, CCStroke etc. with a total of 15 attributes per patient. To simulate the verically partioned data setting, each party takes an arbitrary number of attributes from the scheme. Then, the protocol begins.

The total computation cost for this protocol is dependent on the number of present parties r, the number of common entities n and the operations on encrypted data, which we will represent as m bits of communication. The $secureCompare$ part of the protocol requires a circuit with a linear number of gates in regards to the compared numbers, which are distances. So, we can approximate the communication cost for each $secureCompare$ call as $O(m)$ bits.

In the $closestCluster$ algorithm we have communication at step 4-5 which require $r-1$ rounds of the $permutationAlgorithm$, while steps 8-9 requires $r-2$ rounds of communication of $(r-2)*k*m$ bits. Steps 14-16 require $r-1$ calls of $secureCompare$ and steps 20-21 require $r*k$ bits of communication between P_1 and all other parties. Thus, the total cost is $2(r-1)+r-2+(k-1)*const \approx 3r+const*k = O(r+k)$ rounds and $2k*m*(r-1)+k*m*(r-2)+(k-1)*const \approx O(kr)$ bits.

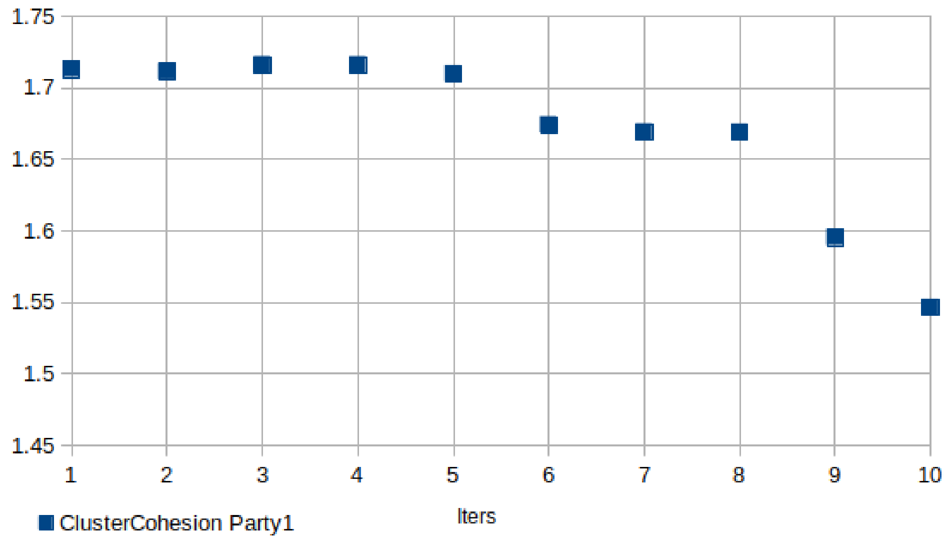
The following tests were ran on a set of 650 entries, with 10 iterations and 4 participating parties, each having 5 attributes. For example, P_0 has the first 5 attributes, P_1 has the next 5 and so on. The observed communication and operations are illustrated in table 3.1

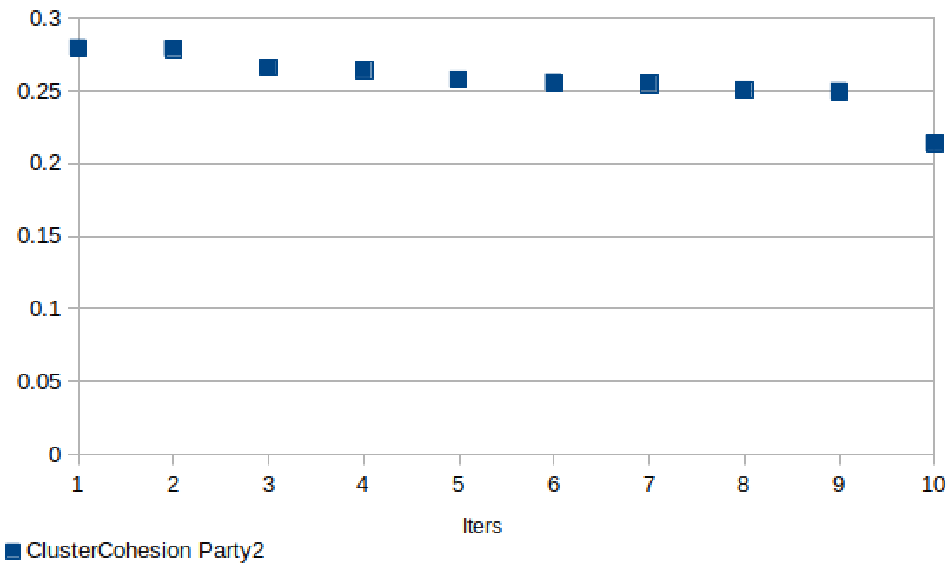
Clusters	SecureCompare operations	Addition operations
3	3676	33967
4	5402	47016
5	7234	60076
10	23674	188064
12	27352	258083

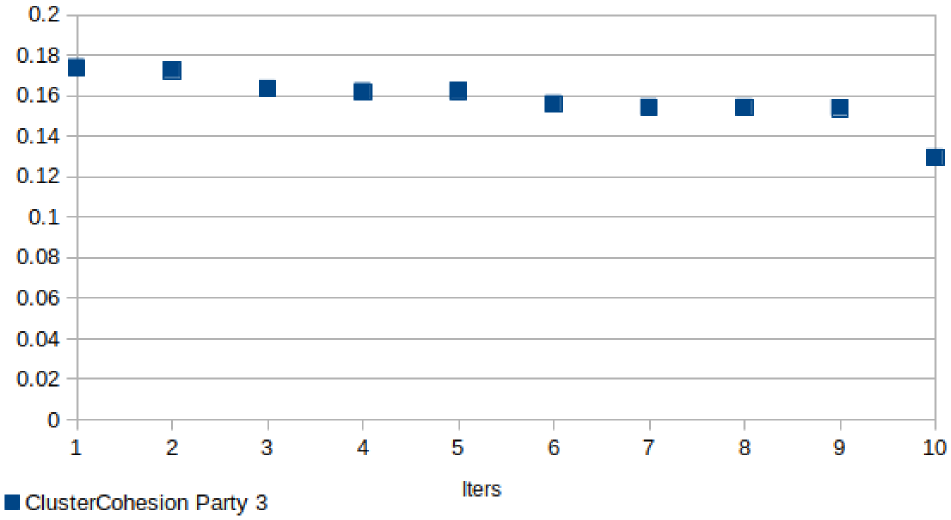
Table 3.1: Communication between parties

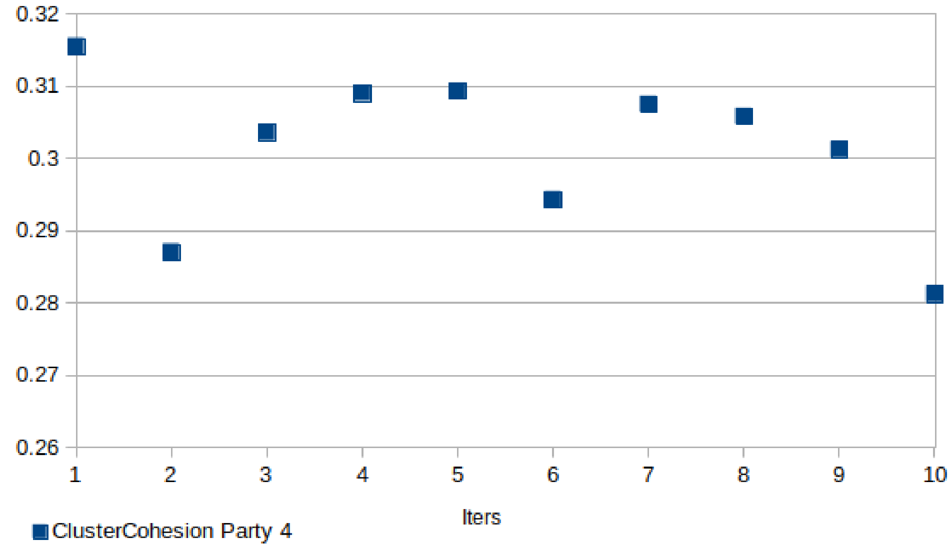
We measured the *cluster cohesion* and *cluster separation* for 10 iterations of this algorithm, with 4 participating parties and 650 entries and 5 clusters. The *cluster cohesion* measures how close are the points in each resulted cluster and it is defined as follows:

$$\sum_{i=1}^{k} \sum_{x \in Clusters[i]} (x - \mu_i)^2$$









As we can see, the general trend is to minimize the cluster cohesion, which signals a good clustering composition through the course of the algorithm's run.

Conclusions

The work made in this thesis describes the current state of art of the work done in the direction of secure multiparty computation. The protocols for SMC accomplished important results; it was shown that any function can be modeled as a generic boolean circuit [GMW87] that can always be evaluated in a secure manner in presence of semi-honest adversaries. Since then, a lot of attention has been drawn to further extend and optimize these protocols.

We made use of different SMC primitives in order to implement a privacy-preserving data-mining algorithm; there are currently several open-source solutions for this, including ([Bog+12], [GC16]), none of which operate on vertically partitioned data like ours. Still, privacy-preserving clustering remains an open problem and it will be considered for the future research.

Bibliography

[Yao82] Andrew Chi-Chih Yao. “Protocols for Secure Computations (Extended Abstract)”. In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. 1982, pp. 160–164. DOI: `10.1109/SFCS.1982.38`.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. 1987, pp. 218–229. DOI: `10.1145/28395.28420`.

[VC03] Jaideep Vaidya and Chris Clifton. “Privacy-preserving *k*-means clustering over vertically partitioned data”. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*. 2003, pp. 206–215. DOI: `10.1145/956750.956776`.

[Bog+12] Dan Bogdanov et al. “High-performance secure multi-party computation for data mining applications”. In: *Int. J. Inf. Sec.* 11.6 (2012), pp. 403–418. DOI: `10.1007/s10207-012-0177-2`.

[GC16] Zakaria Gheid and Yacine Challal. “Efficient and Privacy-Preserving k-Means Clustering for Big Data Mining”. In: *2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, August 23-26, 2016*. 2016, pp. 791–798. DOI: `10.1109/TrustCom.2016.0140`.

[Țip06] Ferucio Laurențiu Țiplea. “Fundamentele algebrice ale informaticii”. In: Ed. Polirom, 2006.

[AM69] Michael Francis Atiyah and I. G. MacDonald. *Introduction to commutative algebra*. Addison-Wesley-Longman, 1969. ISBN: 978-0-201-40751-8.

[CDN15] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. ISBN: 9781107043053. URL: `http://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB%5C&isbn=9781107043053`.

[LP09] Yehuda Lindell and Benny Pinkas. "A Proof of Security of Yao's Protocol for Two-Party Computation". In: *J. Cryptology* 22.2 (2009), pp. 161–188. DOI: `10.1007/s00145-008-9036-8`.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: `10.1145/359168.359176`.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. "A Randomized Protocol for Signing Contracts". In: *Commun. ACM* 28.6 (1985), pp. 637–647. DOI: `10.1145/3812.3818`.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (1978), pp. 120–126. DOI: `10.1145/359340.359342`.

[Has+19] Marcella Hastings et al. "SoK: General Purpose Compilers for Secure Multi-Party Computation". In: *SoK: General Purpose Compilers for Secure Multi-Party Computation*. IEEE. 2019.

[Rab05] Michael O. Rabin. "How To Exchange Secrets with Oblivious Transfer". In: *IACR Cryptology ePrint Archive* 2005 (2005), p. 187. URL: `http://eprint.iacr.org/2005/187`.

[Kil88] Joe Kilian. "Founding Cryptography on Oblivious Transfer". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. 1988, pp. 20–31. DOI: `10.1145/62212.62215`.

[Cré87] Claude Crépeau. "Equivalence Between Two Flavours of Oblivious Transfers". In: *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*. 1987, pp. 350–354. DOI: `10.1007/3-540-48184-2\_30`.

[Ish+03] Yuval Ishai et al. “Extending Oblivious Transfers Efficiently”. In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. 2003, pp. 145–161. DOI: 10.1007/978-3-540-45146-4_9.

[Pai99a] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[Gam84] Taher El Gamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2.

[LNV11] Kristin E. Lauter, Michael Naehrig, and Vinod Vaikuntanathan. “Can Homomorphic Encryption be Practical?” In: *IACR Cryptology ePrint Archive* 2011 (2011), p. 405. URL: http://eprint.iacr.org/2011/405.

[Lin09] Yehuda Lindell. “Secure Computation for Privacy Preserving Data Mining”. In: *Encyclopedia of Data Warehousing and Mining, Second Edition (4 Volumes)*. 2009, pp. 1747–1752. URL: http://www.igi-global.com/Bookstore/Chapter.aspx?TitleId=11054.

[IG03] Ioannis Ioannidis and Ananth Grama. “An Efficient Protocol for Yao's Millionaires' Problem”. In: *36th Hawaii International Conference on System Sciences (HICSS-36 2003), CD-ROM / Abstracts Proceedings, January 6-9, 2003, Big Island, HI, USA*. 2003, p. 205. DOI: 10.1109/HICSS.2003.1174464.

[LT05] Hsiao-Ying Lin and Wen-Guey Tzeng. “An Efficient Solution to the Millionaires' Problem Based on Homomorphic Encryption”. In: *Applied Cryptography and Network Security*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 456–466. ISBN: 978-3-540-31542-1.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. “Foundations of garbled circuits”. In: *the ACM Conference on Computer and Communications*

Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012. 2012, pp. 784–796. DOI: 10.1145/2382196.2382279.

[Kur14] Kaoru Kurosawa. "Garbled Searchable Symmetric Encryption". In: *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*. 2014, pp. 234–251. DOI: 10.1007/978-3-662-45472-5_15.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. ISBN: 0-521-83084-2.

[Bar+14] Joshua Baron et al. "How to withstand mobile virus attacks, revisited". In: *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*. 2014, pp. 293–302. DOI: 10.1145/2611462.2611474.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[BF98] Paul S. Bradley and Usama M. Fayyad. "Refining Initial Points for K-Means Clustering". In: *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*. 1998, pp. 91–99.

[YPS10] Madhu Yedla, Srinivasa Rao Pathakota, and TM Srinivasa. "Enhancing K-means clustering algorithm with improved initial center". In: *International Journal of computer science and information technologies* 1.2 (2010), pp. 121–125.

[Hin19] Pieter Hintjens. *PyZMQ: Python bindings for zeromq*. https://github.com/zeromq/pyzmq. 2019.

[Pai99b] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[19] *Partially Homomorphic Encryption using the Paillier crypto system*. https://python-paillier.readthedocs.io/en/develop. Accessed: 2019-07-01. 2019.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. *EMP-toolkit: Efficient MultiParty computation toolkit*. https://github.com/emp-toolkit. 2016.