Secure Multi-Party Computations On Graphs

Applied Mathematics M.Sc. Thesis

By:

Szilvia Lestyán

Advisor:

András Lukács

Department of Computer Science

Eötvös Loránd University

Faculty of Natural Sciences

2016

Acknowledgements

I would like to thank my friends and colleagues for the listening and thoughtful discussions, not limited to the scope of this present work. I would also like to acknowledge the patience and understanding of my family. Finally I would like to thank my advisor, András Lukács, for his interesting ideas and guidance.

Budapest, 01. January, 2016

Szilvia Lestyán

Contents

1 Introduction **4**

2 Preliminaries **6**

2.1 Distributed Algorithms 6

2.2 Privacy-Preserving Data Mining 8

2.2.1 Models of PPDM 9

2.3 Secure Multi-Party Computations 11

2.3.1 Adversarial power 14

2.3.2 Definitions of security 17

2.3.3 Techniques for building secure multiparty computation protocols 19

2.4 Applied model 24

3 Algorithms **27**

3.1 Secure BFS 27

3.2 Leader election 29

3.3 Secure Min/Max Search on graphs 31

3.4 Secure Sum on graphs 33

3.5 Secure Vertex Coloring 36

4 Future Challenges **42**

Summary

The importance of secure multi-party computations has risen in the past decade, nowadays we are not only considering secret sharing in the original two-party environment, but trying to extend it on arbitrary numbers of participants. Many combinations exist on the setting of the model, i.e. the distribution of the secret, the topology of the parties, the available connection among them etc. This thesis focuses on the graph model as in distributed computing. The parties correspond to the nodes (or processors), one party can do any local computations, can communicate only directly with its neighbours, and knows nothing about the structure of the global graph or or the data held by other nodes. The communication has to proceed privately, i.e. parties engaging in a protocol cannot get hold of any information besides its own input and the output of the algorithm. To respect this principle cryptographic functions are applied which are known to be secure. In this paper new protocols are created for the following problems: breath-first-search, leader election, secure sum, minimum and maximum search of all inputs and also graph colouring. For the latter four problems the below detailed algorithms are completely new, moreover the approach of the problems, i.e. the combination of secure computation and distributed computing are also unprecedented.

Chapter 1

Introduction

Data mining is a technology that blends traditional data analysis methods with sophisticated algorithms for processing large volumes of data [4]. It has also opened up exciting opportunities for exploring and analyzing new types of data and for analyzing old types of data in new ways.

Data mining techniques can be used to support a wide range of business intelligence applications such as customer profiling, targeted marketing, workflow management, store layout, and fraud detection. It can also help retailers answer important business questions such as "Who are the most profitable customers?" "What products can be cross-sold or up-sold?" and "What is the revenue outlook of the company for next year?". Researchers in medicine, science, and engineering are rapidly accumulating data that is key to important new discoveries.

Data mining is the process of automatically highlighting useful information in large data repositories. Data mining techniques are deployed to scour large databases in order to find novel and useful patterns that might otherwise remain unknown. They also provide capabilities to predict the outcome of a future observation.

Techniques from high performance (parallel) computing are often important in addressing the massive size of some data sets. Distributed techniques can also help address the issue of size and are essential when the data cannot be gathered in one location.

Confidentiality issues in data mining. A key problem that arises in any en masse collection of data is that of confidentiality. The need for privacy is sometimes due to law (e.g., for medical databases) or can be motivated by business interests. However, there are situations where the sharing of data can lead to mutual gain. A key application domain of large databases today is research, whether it be scientific, or economic and market oriented. Thus, for example, the medical field has much to gain by pooling data for research; as can even competing businesses with mutual interests. Despite the potential gain, this is often not possible due to the confidentiality issues which arise.

The concept of Secure Multiparty Computation was introduced in [18] and has been proved that there is a secure multi-party computation solution for any polynomial function [19]. Thus, in the case of private data mining, more efficient solutions are required. Secure two party computation was first investigated by Yao, and was later generalized to multiparty computation [1]. These works all use a similar methodology: the function f to be computed is first represented as a combinatorial circuit, and then the parties run a short protocol for every gate in the circuit. While this approach is appealing in its generality and simplicity, the protocols it generates depend on the size of the circuit. This size depends on the size of the input (which might be huge as in a data mining application), and on the complexity of expressing f as a circuit (for example, a naive multiplication circuit is quadratic in the size of its inputs). We stress that secure two-party computation of small circuits with small inputs may be practical using some protocols.

Chapter 2

Preliminaries

2.1 Distributed Algorithms

Distributed algorithms [5,8,10] are the cornerstone of this thesis, but not on their own, it is combined with secure computations to create new algorithms in a mixed context [12,15].

- **Processes** A distributed system is made up of a collection of computing units, each one abstracted through the notion of a process. The processes are assumed to cooperate on a common goal, which means that they exchange information in one way or another. The set of processes is static. It is composed of n processes and denoted $\Pi = \{p_1, ..., p_n\}$, where each $p_i, 1 \leq i \leq n$, represents a distinct process. Each process p_i is sequential, i.e., it executes one step at a time. The integer i denotes the index of process p_i , i.e., the way an external observer can distinguish processes. It is always assumed that each process p_i has its own identity, denoted id_i; then p_i knows id_i (in a lot of cases - but not always - $id_i = i$).

- **Communication Medium** The processes communicate by sending and receiving messages through channels. Each channel is assumed to be reliable (it does not create, modify, or duplicate messages).

- **Structural View** It follows from the previous definitions that, from a structural point of view, a distributed system can be represented by a connected undirected graph $G = (\Pi, C)$ (where C denotes the set of channels).

- **Distributed Algorithm** A distributed algorithm is a collection of n automata, one per process. An automaton describes the sequence of steps executed by the corresponding process. In addition to the power of a Turing machine, an automaton is enriched with two communication operations which allows it to send a message on a channel or receive a message on any channel. The operations are *send*() and *receive*().

- **Synchronous Algorithm** A distributed synchronous algorithm is an algorithm designed to be executed on a synchronous distributed system. The progress of such a system is governed by an external global clock, and the processes collectively execute a sequence of rounds, each round corresponding to a value of the global clock. During a round, a process sends at most one message to each of its neighbors. The fundamental property of a synchronous system is that a message sent by a process during a round r is received by its destination process during the very same round r. Hence, when a process proceeds to the round $r + 1$, it has received (and processed) all the messages which have been sent to it during round r, and it knows that the same is true for any process.

- **Asynchronous Algorithm** A distributed asynchronous algorithm is an algorithm designed to be executed on an asynchronous distributed system. In such a system, there is no notion of an external time. That is why asynchronous systems are sometimes called time-free systems. In an asynchronous algorithm, the progress of a process is ensured by its own computation and the messages it receives. When a process receives a message, it processes the message and, according to its local algorithm, possibly sends messages to its neighbors. A process processes one message at a time. This means that the processing of a message cannot be interrupted by the arrival of another message. When a message arrives, it is added to the input buffer of the receiving process. It will be processes after all the messages that precede it in this buffer have been processed.

- **Initial Knowledge of a Process** When solving a problem in a synchronous/asynchronous system, a process is characterized by its input parameters (which are related to the problem to solve) and its initial knowledge of its environment. This knowledge concerns its identity, the total number n of processes)which is usually unknown at the beginning), the identity of its neighbors, the structure of the communication graph, etc.

2.2 Privacy-Preserving Data Mining

Privacy-preserving data mining considers the problem of running data mining algorithms on confidential data that is not supposed to be revealed even to the party running the algorithm [9,11]. There are two classic settings of privacy-preserving data mining (although these are by no means the only ones [2]). In the first, the data is divided among two or more different parties; the aim being to run a data mining algorithm on the union of the parties' databases without allowing any party to view another individual's private data. In the second, some statistical data that is to be released (so that it can be used for research using statistics and/or data mining) may contain confidential data; hence, it is first modified so that (a) the data does not compromise anyone's privacy, and (b) it is still possible to obtain meaningful results by running data mining algorithms on the modified data set. In this paper, we will mainly refer to scenarios of the first type.
A classical example of a privacy-preserving data mining problem of the first type occurs in the field of medical research. Consider the case of a number of different hospitals that wish to jointly mine their patient data for the purpose of medical research. Furthermore, let us assume that privacy policy

and law prevents these hospitals from ever pooling their data or revealing it to each other, due to the confidentiality of patient records. In such cases, classical data mining solutions cannot be used. Rather, it is necessary to find a solution that enables the hospitals to compute the desired data mining algorithm on the union of their databases, without ever pooling or revealing their data. Privacy-preserving data mining solutions have the property that the only information (provably) learned by the different hospitals is the output of the data mining algorithm.
This problem, whereby different organizations cannot directly share or pool their databases, yet must nevertheless carry out joint research via data mining, is quite common.
Another example relates to data that is held by governments. In the late 1990s, the Canadian Government held a vast federal database that pooled citizen data from a number of different government ministries; this database was officially called the Longitudinal Labor Force File, but became known as the "big brother" database. The aim of the database was to implement governmental research that would arguably improve the services received by citizens. However, due to privacy concerns and public outcry, the database was dismantled, thereby preventing such "essential research" from being carried out. This is another example where privacy-preserving data mining could be used to balance real privacy concerns with the needs of governments to carry out important research.

2.2.1 Models of PPDM

In the study of privacy-preserving data mining (PPDM), there are mainly four models as follows [1]:

1. **Trust Third Party Model:** The goal standard for security is the assumption that we have a trusted third party to whom we can give all data. The third party performs the computation and delivers only the results - except for the third party, it is clear that nobody learns

anything not inferable from its own input and the results. The goal of secure protocols is to reach this same level of privacy preservation, without the problem of finding a third party that everyone trusts.

2. **Semi-honest Model:** In the semi-honest model, every party follows the rules of the protocol using its correct input, but after the protocol is free to use whatever it sees during execution of the protocol to compromise security.

3. **Malicious Model:** In the malicious model, no restrictions are placed on any of the participants. Thus any party is completely free to indulge in whatever actions it pleases. In general, it is quite difficult to develop efficient protocols that are still valid under the malicious model. However, the semi-honest model does not provide sufficient protection for many applications.

4. **Other Models** - Incentive Compatibility: While the semi-honest and malicious models have been well researched in the cryptographic community, other models outside the purview of cryptography are possible. One example is the interesting economic notion of incentive compatibility. A protocol is incentive compatible if it can be shown that a cheating party is either caught or else suffers an economic loss. Under the rational model of economics, this would serve to ensure that parties do not have any advantage by cheating. Of course, in an irrational model, this would not work.

I remark, that in the "real world", there is no external party that can be trusted by all parties, so the Trust Third Party Model is an ideal model.

2.3 Secure Multi-Party Computations

This problem deals with a setting where a set of parties with private inputs wishes to jointly compute some function of their inputs [7]. Loosely speaking, this joint computation should have the property that the parties learn the correct output and nothing else, even if some of the parties maliciously collude to obtain more information. Clearly, a protocol that provides this guarantee can be used to solve privacy-preserving data mining problems of the type discussed above.
Distributed computing considers the scenario where a number of distinct, yet connected, computing devices (or parties) wish to carry out a joint computation of some function. For example, these devices may be servers who hold a distributed database system, and the function to be computed may be a database update of some kind. The aim of secure multiparty computation is to enable parties to carry out such distributed computing tasks in a secure manner. Whereas distributed computing classically deals with questions of computing under the threat of machine crashes and other inadvertent faults, secure multiparty computation is concerned with the possibility of deliberately malicious behavior by some adversarial entity. That is, it is assumed that a protocol execution may come under "attack" by an external entity, or even by a subset of the participating parties. The aim of this attack may be to learn private information or cause the result of the computation to be incorrect. Thus, two important requirements on any secure computation protocol are privacy and correctness. The privacy requirement states that nothing should be learned beyond what is absolutely necessary; more exactly, parties should learn their output and nothing else. The correctness requirement states that each party should receive its correct output. Therefore, the adversary must not be able to cause the result of the computation to deviate from the function that the parties had set out to compute.
The setting of secure multiparty computation encompasses tasks as simple as coin-tossing and broadcast, and as complex as electronic voting, electronic auctions, electronic cash schemes, contract signing, anonymous transactions,

and private information retrieval schemes. Consider for a moment the tasks of voting and auctions. The privacy requirement for an election protocol ensures that no parties learn anything about the individual votes of other parties; the correctness requirement ensures that no coalition of parties has the ability to influence the outcome of the election beyond simply voting for their preferred candidate. Likewise, in an auction protocol, the privacy requirement ensures that only the winning bid is revealed (if this is desired); the correctness requirement ensures that the highest bidder is indeed the winning party (and so the auctioneer, or any other party, cannot bias the outcome). Due to its generality, the setting of secure multiparty computation can model almost every cryptographic problem.

Security in multiparty computation

As we have mentioned above, the model that we consider is one where an adversarial entity controls some subset of the parties and wishes to attack the protocol execution. The parties under the control of the adversary are called corrupted, and follow the adversary's instructions. Secure protocols should withstand any adversarial attack (the exact power of the adversary will be discussed later). In order to formally claim and prove that a protocol is secure, a precise definition of security for multiparty computation is required. A number of different definitions have been proposed and these definitions aim to ensure a number of important security properties that are general enough to capture most (if not all) multiparty computation tasks [1]. The most central of these properties:

1. **Privacy:** No party should learn anything more than its prescribed output. In particular, the only information that should be learned about other parties' inputs is what can be derived from the output itself. For example, in an auction where the only bid revealed is that of the highest bidder, it is clearly possible to derive that all other bids

were lower than the winning bid. However, this should be the only information revealed about the losing bids.

2. **Correctness:** Each party is guaranteed that the output that it receives is correct. To continue with the example of an auction, this implies that the party with the highest bid is guaranteed to win, and no party including the auctioneer can alter this.

3. **Independence of Inputs:** Corrupted parties must choose their inputs independently of the honest parties' inputs. This property is crucial in a sealed auction, where bids are kept secret and parties must fix their bids independently of others. Note that independence of inputs is not implied by privacy. For example, it may be possible to generate a higher bid without knowing the value of the original one. Such an attack can actually be carried out on some encryption schemes (i.e., given an encryption of 100 dollars, it is possible to generate a valid encryption of \$101, without knowing the original encrypted value).

4. **Guaranteed Output Delivery:** Corrupted parties should not be able to prevent honest parties from receiving their output. In other words, the adversary should not be able to disrupt the computation by carrying out a "denial of service" attack.

5. **Fairness:** Corrupted parties should receive their outputs if and only if the honest parties also receive their outputs. The scenario where a corrupted party obtains output and an honest party does not should not be allowed to occur. This property can be crucial, for example, in the case of contract signing. Specifically, it would be very problematic if the corrupted party received the signed contract and the honest party did not.

2.3.1 Adversarial power

The above informal definition of security omits one very important issue: the power of the adversary that attacks a protocol execution. As we have mentioned, the adversary controls a subset of the participating parties in the protocol. However, we have not described the corruption strategy (i.e., when or how parties come under the "control" of the adversary), the allowed adversarial behavior (i.e., does the adversary passively gather information or can it instruct the corrupted parties to act maliciously), and what complexity the adversary is assumed to be (i.e., is it polynomial-time or computationally unbounded). Lets now view the main types of adversaries that have been considered:
Corruption strategy: The corruption strategy deals with the question of when and how parties are corrupted. There are two main models:

1. **Static corruption model:** In this model, the adversary is given a fixed set of parties whom it controls. Honest parties remain honest throughout, while corrupted parties remain corrupted.

2. **Adaptive corruption model:** Rather than having a fixed set of corrupted parties, adaptive adversaries are given the capability of corrupting parties during the computation. The choice of who to corrupt and when can be arbitrarily decided by the adversary and may depend on its view of the execution; for this reason, it is called adaptive. This strategy models the threat of an external "hacker" breaking into a machine during an execution. We note that in this model, once a party is corrupted, it remains corrupted from that point on.

Allowed adversarial behavior

There are two main types of adversaries:

1. **Semi-honest adversaries:** In semi-honest adversarial model, it correctly follows the protocol specification, yet attempts to learn addi-

tional information by analyzing the transcript of messages received during the execution. This is a rather weak adversarial model. However, there are some settings where it can realistically model the threats to the system. Semi-honest adversaries are also called "honest-but-curious" and "passive".

2. **Malicious adversaries:** In malicious adversarial model, a party For example, consider the interaction between different intelligence agencies. For security purposes, these agencies cannot allow each other free access to their confidential information; if they did, then a single mole in a single agency would have access to an overwhelming number of sources. Nevertheless, as we all know, homeland security also mandates the sharing of information! It is much more likely that suspicious behavior would be detected if the different agencies were able to run data mining algorithms on their combined data.may arbitrarily deviate from the protocol specification. In general, providing security in the presence of malicious adversaries is preferred, as it ensures that no adversarial attack can succeed. Malicious adversaries are also called "active". We remark that although the semi-honest adversarial model is far weaker than the malicious model, it is often a realistic one. This is because deviating from a specified program which may be buried in a complex application is a non-trivial task.

Complexity

Finally, we consider the assumed computational complexity of the adversary. As above, there are two categories here:

1. **Polynomial-time:** The adversary is allowed to run in (probabilistic) polynomial-time (and sometimes, expected polynomial-time). The specific computational model used differs, depending on whether the adversary is uniform (in which case, it is a probabilistic polynomial-time Turing machine) or non-uniform (in which case, it is modeled

by a polynomial-size family of circuits). We remark that probabilistic polynomial-time is the standard notion of "feasible" computation; any attack that cannot be carried out in polynomial-time is not a threat in real life.

2. **Computationally unbounded:** In this model, the adversary has no computational limits whatsoever.

The above distinction regarding the complexity of the adversary yields two very different models for secure computation: the information-theoretic model and the computational model. In the information-theoretic setting, the adversary is not bound to any complexity class (and in particular, is not assumed to run in polynomial-time). Therefore, results in this model hold unconditionally and do not rely on any complexity or cryptographic assumptions. The only assumption used is that parties are connected via ideally private channels (i.e., it is assumed that the adversary cannot eavesdrop or interfere with the communication between honest parties). By contrast, in the computational setting the adversary is assumed to run in polynomial-time. Results in this model typically assume cryptographic assumptions, such as the existence of trapdoor permutations. These are assumptions on the hardness of solving some problem (e.g., factoring large integers) whose hardness has not actually been proven but is widely conjectured. Note that it is not necessary here to assume that the parties have access to ideally private channels, because such channels can be implemented using public-key encryption. However, it is assumed that the communication channels between parties are authenticated; that is, if two honest parties communicate, then the adversary can eavesdrop but cannot modify any message that is sent. Such authentication can be achieved using digital signatures and a public-key infrastructure.

2.3.2 Definitions of security

Cryptographic Preliminaries

First without establishing a mathematical definition we can say that an encryption scheme is secure if no adversary can compute any function of the plaintext from the ciphertext, eg. the ciphertext reveals nothing about the underlying plaintext, we can also say that the distributions over messages and ciphertexts are independent [6]. Formally:

Definition 2.3.1 *An encryption scheme (Gen, Enc, Dec) over a message space M is perfectly secret if for every probability distribution over M, every message $m \in M$, and every ciphertext $c \in C$ for which $Pr[C = c] > 0$:*

$$Pr[M = m|C = c] = Pr[M = m].$$

There are cryptographic schemes that can be mathematically proven secure (with respect to some particular definition of security), even when the adversary has unlimited computational power. Such schemes are called **information-theoretically secure**, or perfectly secure, because their security is due to the fact that the adversary simply does not have enough "information" to succeed in its attack, regardless of the adversary's computational power. In particular, as we have discussed, the ciphertext in a perfectly-secret encryption scheme does not contain any information about the plaintext (assuming the key is unknown).
Information-theoretic security stands in stark contrast to **computational security** that is the aim of most modern cryptographic constructions. Restricting ourselves to the case of private-key encryption (though everything we say applies more generally), modern encryption schemes have the property that they can be broken given enough time and computation, and so they do not satisfy the above definition. Nevertheless, under certain assumptions, the amount of computation needed to break these encryption schemes would take more than many lifetimes to carry out even using the fastest available

supercomputers. For all practical purposes, this level of security suffices.

The computational approach incorporates two relaxations of the notion of perfect security:

1. Security is only preserved against efficient adversaries, and

2. Adversaries can potentially succeed with some very small probability.

We denote the security parameter by n (n here denotes the security parameter and x is the inputs to the protocol); essentially, this parameter determines the length of cryptographic keys (or more exactly, the length of input needed to solve some hard problem so that real-world adversaries cannot break the problem in a reasonable amount of time). We say that a function $\mu(.)$ is **negligible** in n (or just negligible) if for every positive polynomial $p(.)$ there exists an integer N such that for all $n > N$ it holds that $\mu(n) < 1/p(n)$. Note that an event that happens with negligible probability happens so infrequently that we can effectively dismiss it.
All parties, including the adversary, run in time that is polynomial in n. We assume that each party has a "security parameter tape" that is initialized to the string of n ones, denoted 1^n; the parties then run in time that is polynomial in the input written on that tape.
We define efficient computation as that which can be carried out in probabilistic polynomial time. An algorithm A is said to run in polynomial time if there exists a polynomial $p(.)$ such that, for every input $x \in \{0,1\}^*$, the computation of $A(x)$ terminates within at most $p(||x||)$ steps (here, $||x||$ denotes the length of the string x). A scheme is secure if every *ppt* (probabilistic polynomial time) adversary succeeds in breaking the scheme with only negligible probability.
A typical proof of security for a cryptographic scheme might show that any adversary running in time $p(n)$ succeeds with probability at most $\frac{1}{p(n)^{2n}}$. This implies that the scheme is (asymptotically) secure, since for any polynomial $p(n)$, the function $\frac{1}{p(n)^{2n}}$ is eventually smaller than any inverse-polynomial in n. A scheme is secure if for every probabilistic polynomial-time adversary

"A" carrying out an attack of some specified type, the probability that A succeeds in this attack (where success is also well defined) is negligible.

Let $X(n, x)$ and $Y(n, x)$ be random variables indexed by n and x, and let $X = \{X(n, x)\}_{n \in N, x \in \{0,1\}^*}$ and $Y = \{Y(n, x)_{n \in N, x \in \{0,1\}^*}\}$ be distribution ensembles. We say that these two random variables are computationally indistinguishable if no algorithm running in polynomial-time can tell them apart (except with negligible probability). More precisely, we say that X and Y are **computationally indistinguishable**, denoted $X \equiv Y$, if for every non-uniform polynomial-time distinguisher D there exists a function $\mu(.)$ that is negligible in n, such that for every $a \in \{0, 1\}^*$,

$$\|Pr[D(X(n, x)) = 1] - Pr[D(Y(n, x)) = 1]\| < \mu(n).$$

Thus, if X and Y are indistinguishable, it holds that for every efficient distinguisher D and for every positive polynomial $p(n)$, there exists an N such that for all $n > N$ it holds that D cannot distinguish between the two with probability better than $1/p(n)$. Therefore, X and Y are the same for all intents and purposes. Typically, the distributions X and Y will denote the output vectors of the parties in real and ideal executions, respectively. In this case, x denotes the parties' inputs. (The outputs of the parties are modeled as random variables since the operation of the parties is typically probabilistic, depending on random coin tosses -or random inputs- used by the parties.)

2.3.3 Techniques for building secure multiparty computation protocols

Homomorphic Encryption:

A homomorphic encryption scheme is an encryption scheme which allows certain algebraic operations to be carried out on the encrypted plaintext, by applying an efficient operation to the corresponding ciphertext. In particular, we will be interested in additively homomorphic encryption schemes

where the message space is a ring (or, more commonly, a field). There exists an efficient algorithm $+_{pk}$ whose input is the public key of the encryption scheme and two ciphertexts, and whose output is $E_{pk}(m_1) +_{pk} E_{pk}(m_2) = E_{pk}(m_1 + m_2)$. (Namely, it is easy to compute, given the public key alone and the encryption of the sum of the plaintexts of two ciphertexts.)
An efficient implementation of an additive homomorphic encryption scheme with semantic security was given by Paillier. In this cryptosystem, the encryption of a plaintext from $[1, N]$, where N is an RSA modulus, requires two exponentiations modulo N^2. Decryption requires a single exponentiation. The Damgard-Jurik cryptosystem is a generalization of the Paillier cryptosystem which encrypts messages from the range $[1, N^s]$ using computations modulo N^s+1, where N is an RSA modulus and s a natural number. It enables more efficient encryption of larger plaintexts than Paillier's cryptosystem (which corresponds to the case $s = 1$). The security of both schemes is based on the decisional composite residuosity assumption:
The **decisional composite residuosity problem**, roughly speaking, is to distinguish a random element of $\mathbb{Z}^*_{N^2}$ from a random element of $Res(N^2)$. Formally, let $GenModulus$ be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$, and p and q are n-bit primes (except with probability negligible in n).Then:

Definition 2.3.2 *We say the decisional composite residuosity problem is hard relative to GenModulus if for all probabilistic, polynomial-time algorithms A there exists a negligible function negl such that*

$$|Pr[\mathcal{A}(N, [r^N mod N^2]) = 1] - Pr[\mathcal{A}(N, r) = 1]| \leq negl(n),$$

where in each case the probabilities are taken over the experiment in which $GenModulus(1^n)$ *outputs* (N, p, q)*, and then a random* $r \leftarrow \mathcal{Z}^*_{N^2}$ *is chosen.*

Secure Sum Protocol

Each participant holds onto a number of their own, and they would like to privately compute the sum of their inputs. One Party, lets say the Main Party, generates a random number R, adds to its local value and sends it to the next party. All participants add their local value to the received number. At last Main Party receives the sum, subtracts R from the result and broadcasts the result [3].

Secure Union Protocol

Let $p \geq 3$ be the number of participants, they all hold onto some sets, denote their respective sets by H_i for $i = 1...p$. We would like to determine the union of all sets without revealing any of them. First every participant has to choose a commutative encryption function. An encryption algorithm is commutative if given encryption keys $K_1, ..., K_n \in K$, for any m in domain M, and for any permutation i, j, the following two equations hold:
(1) $E_{K_{i_1}}(...E_{K_{i_n}}(M)...) = E_{K_{j_1}}(...E_{K_{j_n}}(M)...)$
$\forall M_1, M_2 \in M$ such that $M_1 \neq M_2$ and for given $k, \epsilon < \frac{1}{s^k}$:
(2) $(E_{K_{i_1}}(...E_{K_{i_n}}(M)...) = E_{K_{j_1}}(...E_{K_{j_n}}(M)...)) < \epsilon$
Now all participants encrypt their items one-by-one, and send it to an other party, who encrypts the received encrypted items with his own encryption function and sends it again. They iterate it until each party encrypts the items of the remaining parties. All parties send their r-times encrypted functions to one party (from now on referred to as Main Party), who removes the duplicates. Now this global set is passed around, each site decrypting its items. The union is obtained.

Adversarial Attack Scenarios

We wrap up our general discussion of encryption with a brief discussion of some basic types of attacks against encryption schemes. In order of severity, these are:

- **Ciphertext-only attack:** This is the most basic type of attack and refers to the scenario where the adversary just observes a ciphertext and attempts to determine the plaintext that was encrypted.
- **Known-plaintext attack:** Here, the adversary learns one or more pairs of plaintexts/ciphertexts encrypted under the same key. The aim of the adversary is then to determine the plaintext that was encrypted to give some other ciphertext (for which it does not know the corresponding plaintext).
- **Chosen-plaintext attack:** In this attack, the adversary has the ability to obtain the encryption of any plaintext of its choice. It then attempts to determine the plaintext that was encrypted to give some other ciphertext.
- **Chosen-ciphertext attack:** The final type of attack is one where the adversary is even given the capability to obtain the decryption of any ciphertext of its choice. The adversary's aim, once again, is then to determine the plaintext that was encrypted to give some other ciphertext (whose decryption the adversary is unable to obtain directly).

Security in the Presence of Semi-Honest Adversaries

A multi-party protocol problem is cast by specifying a random process that maps *m-tuples* (do not forget that n is the security parameter) of inputs to *m-tuples* of outputs (one for each party) [1]. We refer to such a process as a $functionality$ and denote it
$f : \{0,1\}^* \times \{0,1\}^* ... \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^* ... \times \{0,1\}^*$, where
$f = (f_1, f_2, ..., f_m)$. That is, for every pair of inputs $x_1, x_2, ..., x_m \in \{0,1\}^m$, the output-tuple is a random variable $(f_1(x_1, x_2, ..., x_m), (f_2(x_1, x_2, ..., x_m), ...,$ $(f_m(x_1, x_2, ..., x_m))$ ranging over m-touples of strings. The first party wishes

to obtain $(f_1(x_1, x_2, ..., x_m)$, and the second party wishes to obtain $(f_1(x_1, x_2, ..., x_m)$. We often denote such a functionality by

$$(x_1, x_2, ..., x_m) \mapsto (f_1(x_1, x_2, ..., x_m), (f_2(x_1, x_2, ..., x_m), ..., (f_n(x_1, x_2, ..., x_m)).$$

Intuitively, a protocol is secure if whatever can be computed 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 simulatable given only its input and output. This then implies that the parties learn nothing from the protocol execution itself, as desired.
Let's continue with the following notation:

- Let $(f_1(x_1, x_2, ..., x_m)$ be a probabilistic polynomial-time functionality and let π be an n-party protocol for computing f.
- The view of the i^{th} party $(i \in \{1, 2, .., m\})$ during an execution of π on input $(x_1, x_2, ..., x_m)$ and security parameter n is denoted $view_i^{\pi}(n, x_1, x_2, .., x_m)$ and equals $(1^n, x_i, r^i, m_1^i, ..., m_t^i)$, where r^i equals the contents of the i^{th} party's internal random tape, and m_j^i represents the j^th message that it received.
- The output of the i^{th} party during an execution of π on input $(x_1, x_2, ..., x_m)$ and security parameter n is denoted $output_i^{\pi}(n, x_1, x_2, ..., x_m)$ and can be computed from its own view of the execution. Denote:

$$output^{\pi}(x_1, x_2, ..., x_m) = \\ (output_1^{\pi}(x_1, x_2, ..., x_m), ..., output_n^{\pi}(x_1, x_2, ..., x_m)).$$

Note that $view_i^{\pi}(x_1, x_2, ..., x_m)$ and $output^{\pi}(x_1, x_2, ..., x_m)$ are random variables, with the probability taken over the random tapes of all the parties.

The definition below quantifies only over inputs x_i and $x_j (\forall \{i, j\} \in \{1, ..., m\})$ that are of the same length. Some restriction on input lengths is required, and padding can be used to achieve this restriction.

Definition 2.3.3 *(security w.r.t. semi-honest behavior): Let* $f = (f_1, f_2, ..., f_m)$ *be a functionality. We say that* π *securely computes* f *in the presence of* $static\ semi-honest\ adversaries$ *if there exist probabilistic polynomial-time algorithms* S_i $(i = 1, ..., m)$ *such that for every* $x_i \in \{0,1\}^*$ *where* $|x_i| = |x_j| \forall \{i,j\} \in \{1, ..., m\}$*, we have:*

$$\{(S_i(1^n, x_i, f_i(x_1, x_2, ..., x_m)), f(x_1, x_2, ..., x_m))\}_{n \in N} \equiv$$
$$\{(view_i^\pi(n, x_1, x_2, ..., x_m), output^\pi(n, x_1, x_2, ..., x_m))\}_{n \in N}\ \forall \{i,j\} \in \{1, ..., m\})$$

This equation states 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. This can be seen by the fact that S_i is given x_i and $f(x_1, x_2, ..., x_m)$ and must generate an output that is indistinguishable from the view of P_i in a real execution. We note that it is not enough for the simulator S_i to generate a string indistinguishable from $(view_i^\pi(n, x_1, x_2, ..., x_m)$. Rather, the joint distribution of the simulator's output and the functionality output $f(n, x_1, x_2, ..., x_m)$ must be indistinguishable from $(view_i^\pi(n, x_1, x_2, ..., x_m)$, $output^\pi(x_1, x_2, ..., x_m))$. This is necessary for probabilistic functionalities.

2.4 Applied model

Now that we have learned the possible settings of a multi-party computation, let's see the details of the model applied in this thesis:
Assume that the adversary is $semi-honest$ and *static*. That is, it correctly follows the protocol specification, yet attempts to learn additional information by analyzing the transcript of messages received during the execution. Although the semi-honest adversarial model is far weaker than the malicious model (where a party may arbitrarily deviate from the protocol specification), it is often a realistic one. This is because deviating from a specified program which may be buried in a complex application is a non-trivial task. Semi-honest adversarial behavior also models a scenario in which both parties that participate in the protocol are honest. However, following the protocol

execution, an adversary may obtain a transcript of the protocol execution by breaking into one of the parties' machines.

Let $P_1, P_2, ..., P_m$ be parties owning (large) private databases $D_1, D_2, .., D_m$. The parties wish to apply a function to the joint database $\bigcup D_i$ without revealing any unnecessary information about their individual databases. That is, the only information learned by P_i about D_{-i} (where D_{-i} is any other database except D_i) is that which can be learned from the output of the algorithm, and vice versa. We *do not assume any "trusted" third party* who computes the joint output. (Now we can see that the processes of distributed computing correspond to the parties in this model.)
We also assume that a unique label is given to each party (or we shall say nodes) at the formation of the graph, or at the reformation of the graph, eg. between two algorithms nodes can join or nodes can leave the graph.
We do not assume a peer-to-peer system to be available, eg. it is not necessary for all the parties to be directly connected. (Also it corresponds better to a real life setting.) A channel between two parties is bidirectional (can carry messages in both directions), and between all parties leads a route, moreover let's assume that the channels are *first in first out* (FIFO) which means that the messages received in the order in which they have been sent. This concludes that the topology of the parties corresponds with an *arbitrary strongly connected graph*. In case we were to consider networks we could say that the model is similar to a mesh network (a mesh network is a network topology in which each node relays data for the network, all mesh nodes cooperate in the distribution of data in the network), but this setting is different considering security. The channels between any two parties can be *secure* or *insecure* as well, I will consider both cases at each protocol discussed in the next chapter. A secure channel is a way of transferring data that is resistant to overhearing and tampering. In case of an insecure channel an eavesdropper can catch any message (ciphertext) from any existing channel and try to break it. Each party P_i has a set of neighbors, denoted N_i . According to the

context, this set contains the identities (labels) of these processes. This is the only knowledge a node (participant) can have of the global graph, eg. it cannot "see" anything else but its direct neighbours, it does not even know the total number of participants initially (although it could be the result of a given protocol).
This model is *partially synchronous* (timing-based), i.e. we assume some restrictions on the relative timing of events, but execution is not completely lock-step as it is in the synchronous model. These models are the most realistic, but they are also the most difficult to program. Algorithms designed using knowledge of the timing of event can be efficient, but they can also be fragile in that they will not run correctly if the timing assumptions are violated.

Chapter 3

Algorithms

3.1 Secure BFS

I use the breath-first-search algorithm as a basic framework for communication. Along the edges of the resulting tree proceeds the encrypted message, thus this gives us a unique order in which the computation proceeds.
The distributed version of the BFS algorithm can be found in [8]. The basic idea for this algorithm is the same as for the standard sequential breadth-first-search algorithm:

Algorithm 1 *At any point during execution, there are some nodes that are "marked", initially just i_0, the root. The root sends out a search message at round 1 to all of its neighbours. At any round, if an unmarked node receives a search message, it marks itself and chooses one of the nodes from which the search message has arrived as its parent. At the first round after a process gets marked, it sends a search message to all of its outgoing neighbours.*

There exist a variant where each node learns not only who its parent in the tree is, but also who all of its children are. This is very important because during the following secure computations a node does not want to send all of its neighbours its output (mainly because of secrecy reasons), and also it does not want to wait for incoming messages from those neighbours of whom the particular node is not a parent (the message would never arrive). Thus it is necessary for each node receiving a *search* message to respond to that message with a *child* or *non-child* message, telling the sender whether or not it has been chosen by the recipient as the parent. Moreover we also need to mark the end of the algorithm to let everyone know that the framework is available and desired computations can start along the BFS tree edges. Consequently we need to add extra messages to the algorithm.

Algorithm 2 *The root sends out the search messages as before. When a node receives a search message it instantly sets and notifies its chosen parent with a child message, and sends non-child message to those nodes from which it received a search message but did not choose them as parent. After this the node sends out the search message to the rest of its neighbours. The algorithm continues until we reach a leaf node. A leaf node realizes this by receiving non-child message from all of its children candidates. Now we need to notify all nodes in the graph up to the root that the search has ended. The leaf sends an end message to its parent. A non-leaf node sends the end message to its parent after it received the child, non-child or end message from all of its non-parent neighbours. The algorithm ends when the root is in a position to send the end message*

Security in the presence of semi-honest adversaries:

1. *Privacy:* Complies. No nodes learns anything about the global graph, they learn only their parent and children nodes, i.e. the local output.
2. *Independence of inputs*:Complies trivially.
3. *Output delivery:* Complies trivially.
4. *Correctness:* Complies trivially.
5. *Fairness:* Complies trivially.

Security in the presence of malicious adversaries:
Lets consider the scenario when a node intentionally deviates from the protocol. The only thing it can do without getting noticed is setting several nodes as parents, by this it violates the property of "correctness", because the malicious node would become the child of some nodes of which it should not be, thus corrupts their output. It could cause circles in the output graph, thus it looses its property as a tree and that can cause problems for further computations (e.g.: see below the secure sum algorithm), where the protocol demands a tree.

Complexity:
The time complexity is at most $2 * diam$ rounds, the number of messages transmitted is $3 * |E|$ - one *search* message, one *child* or *non-child message* and one *end* message are transmitted on each edge. Thus the time complexity is $\mathcal{O}(diam)$ and the communication complexity is $\mathcal{O}(|E|)$.

3.2 Leader election

The above algorithm does not consider the question of the root, it starts from the point where this problem is already decided. Of course we can suggest that the nodes were given this information before the formulation of the graph, but let us consider the case when the nodes have to decide on a root, i.e. a leader. The non-secure distributed algorithm is the following [8]:

Algorithm 3 *All nodes who have the need of some common result send out search messages together with their unique label. When a node receives a search message the first time it compares its label to its own, if the new label is greater than its own, it continues the algorithm according to the new label, and saves it, otherwise it does not do anything. Every time it receives a new search message it compares it to its own or the one it saved and if necessary*

updates the BFS algorithm according to the new label. At the end of the algorithm only the largest labeled messages remain and the root of the resulting tree is the node with that particular label.

In order to create a secure version of the above algorithm we need to specify the notion of security. There are two versions of security in this case: (1) no node learns the identity of the root or any other root who started the algorithm, or (2) the identity of the leader is broadcasted at the end of the algorithm, but the identity of other candidates remain hidden. It is clear that the second approach easily follows the first one, but we will also consider the scenario where the elected leader has to (or could) prove that indeed it is the elected node. But before this let us start with the first approach:
The idea of the following secure leader election algorithm is based on the above mentioned distributed one:

Algorithm 4 *During this algorithm every node has to have two unique labels, of which one is public, thus for identification, and the other is private. The algorithm goes as the above one but with the private labels. No neighbour of a node learns who has sent the label, because it is not public, so up on receiving a label it is possible that it came from a neighbour or from anyone else in the graph. Additionally if we want proof from the root that indeed it was its private input. In this case everyone has to send a signature along with its label in the message.*

Security in the presence of semi-honest adversaries:

1. *Privacy:* Partially complies. Upon receiving the labels one node can deduce the total number of nodes in the graph. To eliminate this problem we can send fake values to pad the set of id-s.

2. *Independence of inputs:* Complies trivially.

3. *Output delivery:* Complies trivially.

4. *Correctness:* Complies trivially.

5. *Fairness:* Complies trivially.

Security in the presence of malicious adversaries:
Unfortunately a malicious adversary can alter its output to top any previous label's value, thus the "correctness" property cannot be fulfilled at any time. Furthermore "output delivery" and "fairness" could be breached in some of the cases - e.g. the adversary is the only gateway between some nodes.

Complexity:
All labels pass on each edge once and only once. Thus the complexity is $n|E|$.

3.3 Secure Min/Max Search on graphs

Let's assume that all nodes hold a particular value, which is an answer to a local question (e.g.: degree of the node, latitude or altitude of their position, smallest or highest measured temperature values etc.); we would like to acquire the extrema of them securely.

Algorithm 5 *1. The root invokes its key-generating algorithm, gives it the security parameter 1^n as input. The key-generation algorithm outputs a pair of keys (p_k, s_k) (where p_k is the public, s_k is the secret - or private - key).*

2. The root broadcasts its public key to all nodes in the graph, along with the protocol needed to be completed by the other parties in order to fulfil the query.

3. All non-root nodes invoke their key-generating algorithm, give the security parameter 1^n - which can be derived from the received public *key. The key-generating algorithm outputs only one key k_i, which is a symmetric key.*

4. *All non-root nodes encrypt their values with k_i, and also encrypt k_i with p_k, their output is the following:*
   $c_{i1} \leftarrow Enc_{k_i}(x_i)$ $c_{i2} \leftarrow Enc_{p_k}(k_i)$, $c_i := (c_{i1}, c_{i2})$ *which they send to the root.*

5. *The root decrypts the k_i keys, then decrypts the x_i values:*
   $k_i := Dec_{s_k}(c_{i2})$, $m_i := Dec_{k_i}(c_{i1})) = x_i$.

6. *The root searches the required value - maximum or minimum of all inputs then broadcasts the result.*

Remark: It is highly recommended that a leaf node would generate dummy values and would send it along with its real output, thus its parent would not learn the real size of the underlying subtree.

Security in the presence of semi-honest adversaries:

1. *Privacy:* The root receives all values, thus learns a great deal more than just the prescribed output. We could eliminate this problem using the Yao-protocol between child and parent nodes, but as said earlier, this paper aims to avoid such protocols in order to lower complexity.

2. *Independence of inputs:* No node -except the root- learns any other output during execution, the root could be excluded from the computation in order to keep the independence.

3. *Output delivery:* If the network functions well there should be no problems during the broadcast.

4. *Correctness:* Each party should receive a correct output.

5. *Fairness:* All parties receive their outputs.

Security in the presence of malicious adversaries:
If the root is malicious, unfortunately we cannot guarantee any of the above requirements. It can alter the output and also prevent the last broadcast session, thus no other node would receive the output.
In case an inner node is malicious we can guarantee much more than before. Privacy and independence are given, because of the encryption schema that inner node would learn nothing about others' inputs. The other three requirements cannot be fulfiled for the sub-tree of that particular node. Upon receiving the output it can alter it or stop the execution immediately.

Security in the presence of an evesdroppers:
We know that there are plenty symmetric key encryptions to choose from and they would provide sufficient security in sense of encryption security, i.e. the schema can be secure against ciphertext only attack, chosen chipertext attack etc. depending on the chosen schema.

Complexity:
The number of edges in the BFS tree are $n-1$, thus the number of messages sent is $3*(n-1)$.

3.4 Secure Sum on graphs

Another basic problem among many is the secure summation of particular values owned or locally calculated by each and every node. In the non-secure distributed version we can easily fan-in the sum from leaves to root, i.e. starting from the leaves we sum the values. In case of non-equal inputs this number does not express any information about the underlying sub-tree or the global graph for a node. In the following algorithm I combined this with the earlier mentioned *SecureSumProtocol* and for the encryption I used an additive homomorphic function:

Algorithm 6 *1. The root invokes its key-generating algorithm, gives it* *the security parameter 1^n as input. The key-generation algorithm outputs a pair of homomorphic keys (p_k, s_k).*

2. *The root broadcasts its public key to all nodes in the graph, along with the protocol needed to be completed by the other parties in order to fulfil the query.*

3. *All leaf-nodes generate an r_l random number and do the following computation:*
   $c_1 \leftarrow Enc_{p_k}(r_l)$;
   $c_2 \leftarrow r_l + x_l$;
   $c_l = (c_1, c_2)$.
   That is it encrypts r_l, and sends $Enc_{p_k}(r_l)$ and $r_l + x_l$ (x_l is its private value) to its parent.

4. *No non-leaf node does the same, they only fan-in the two values separately up to the root. They multiply the encrypted values which gives the encrypted sum of the values:*
   $c_1 = c_{11} * c_{12} * ... * c_{1m} = Enc_{p_k}(r_1 + r_2 + ... + r_m)$

5. *The root decrypts the c_1 added random numbers, then subtracts them from the c_2 fanned in result:*
   $R := Dec_{s_k}(c_1)$;
   $M = c_2 - R$.

6. *The root broadcasts the result.*

The above algorithm is NOT secure in the presence of an eavesdropper (capturing the incoming and the outgoing message of one particular node a simple subtractions would give out that nodes private value. Thus we need two different homomorphic encryption keys:

Algorithm 7 1. *The root invokes its key-generating algorithm, gives it the security parameter 1^n as input. The key-generation algorithm outputs TWO pairs of keys (p_{k1}, s_{k1}) and (p_{k2}, s_{k2}).*

2. *The root broadcasts its public keys to all nodes in the graph, along with the protocol needed to be completed by the other parties in order to fulfil the query.*

3. *All leaf-nodes generate an r_l random number and do the following computation:*
   $c_1 \leftarrow Enc_{p_{k1}}(r_l)$;
   $c_2 \leftarrow Enc_{p_{k2}}(r_l + x_l)$;
   $c = (c_1, c_2)$.

4. *All non-leaf nodes fan-in the two values seperately up to the root the following way:*
   $c_1 = (c_{11} * c_{12}) * ... * c_{1m} = Enc_{p_k}(r_1 + r_2 + ... + r_m)\ p \leftarrow Enc_{p_{k2}}(x_l)$;
   $c_2 = (c_{21} * c_{22}) * ... * c_{2m} = Enc_{p_k}(r_1 + x_1 + ... + r_m + x_m)\ c = (c_1, c_2)$.

5. *The root decrypts the c_1 added random numbers, then substracts them from the decrypted c_2 fanned in result:*
   $M = Dec_{s_k}(c_2) - Dec_{s_k}(c_1)$

6. *The root broadcasts the result.*

Security in the presence of semi-honest adversaries:

1. *Privacy:* Complies. No node learns anything more than its prescribed output. The root learns some additional information, but it cant derive the individual inputs.

2. *Independence of inputs:* Complies. All inputs are encrtypted, one node has to break the encryption in order to learn anything.

3. *Output delivery:* Complies trivially.

4. *Correctness:* Complies trivially.

5. *Fairness:* Complies trivially.

Security in the presence of malicious adversaries:
Privacy still complies. If the root is the adversary unfortunately even independence is failed, since the root can alter the result depending on the received sum, moreover with this adversary we cannot guarantee any other requierement. If an inner node is malicious independence complies, but the rest are do not for the subtree of the malicious node.

Complexity The message complexity is $3 * (n - 1)$ again.

3.5 Secure Vertex Coloring

An important graph problem, which is encountered when one has to model application-level problems, concerns vertex coloring [5]. It consists in assigning a value (color) to each vertex such that (a) no two vertices which are neighbors have the same color, and (b) the number of colors is "reasonably small". When the number of colors has to be the smallest possible one, the problem is NP-complete.
Let Δ be the maximal degree of a graph. Remember, that it is always possible to color the vertices of a graph in $\Delta + 1$ colors.
This section presents a distributed algorithm which colors the processes in at most $(\Delta + 1)$ colors in such a way that no two neighbors have the same color. Distributed coloring is encountered in practical problems such as resource allocation or processor scheduling. More generally, distributed coloring algorithms are symmetry breaking algorithms in the sense that they partition the set of processes into subsets (a subset per color) such that no two processes in the same subset are neighbors.

Initial Context: This algorithm assumes that the processes are already colored in $m \geq \Delta + 1$ colors in such a way that no two neighbors have the same color. Let us observe that, from a computability point of view, this is a "no-cost" assumption (because taking $m = n$ and defining the color of a process p_i as its index i trivially satisfies this initial coloring assumption). Differently, taking $m = \Delta + 1$ assumes that the problem is already solved. Hence, the assumption on the value of m is a complexity-related assumption.

Local Variables: Each process p_i manages a local variable $color_i[i]$ which initially contains its initial color, and will contain its final color at the end of the algorithm. A process p_i also manages a local variable $color_i[j]$ for each of its neighbors p_j. As the algorithm is semi-synchronous and round-based, the local variable r_i managed by p_i denotes its current local round number.

Algorithm 8 ***Distributed Coloring:***
The processes proceed in consecutive asynchronous rounds and, at each round, each process synchronizes its progress with its neighbors. As the rounds are semi-synchronous, or timing-based, the round numbers are not given for free by the computation model. They have to be explicitly managed by the processes themselves. Hence, each process p_i manages a local variable r_i that it increases when it starts a new asynchronous round.
The first round is an initial round during which the processes exchange their initial color in order to fill in their local array $color_i[neighbor_i]$. If the processes know the initial colors of their neighbors, this communication round can be suppressed. The processes then execute $m - (\Delta + 1)$ asynchronous rounds.
The processes whose initial color belongs to the set of colors $\{1, ..., \Delta + 1\}$ keep their color forever. The other processes update their colors in order to obtain a color in $\{1, ..., \Delta + 1\}$. To that end, all the processes execute sequentially the rounds $\Delta + 2, ..., m$, considering that each round number corresponds to a given distinct color. During round r, $\Delta + 2 \leq r \leq m$, each process whose initial color is r looks for a new color in $\{1, ..., \Delta + 1\}$ which is not the color of its neighbors and adopts it as its new color. Then, each process exchanges its color with its neighbours before proceeding to the next round. Hence, the round invariant is the following one: When a round r ter-

minates, the processes whose initial colors were in $\{1, ..., r\}$ *(a) have a color in the set* $\{1, ..., \Delta + 1\}$*, and (b) have different colors if they are neighbors.*

Cost The time complexity (counted in number of rounds) is $m - \Delta$ rounds (an initial round plus $m - (\Delta + 1)$ rounds). Each message carries a tag, a color, and possibly a round number which is also a color. As the initial colors are in $\{1, ..., m\}$, the message bit complexity is $O(log_2 m)$.
Finally, during each round, two messages are sent on each channel. The message complexity is consequently $2e(m - \Delta)$, where e denotes the number of channels. It is easy to see that the better the initial process coloring (i.e., the smaller the value of m), the more efficient the algorithm.

Theorem 3.5.1 *Let* $m \geq \Delta + 2$*. The above algorithm is a legal* $(\Delta + 1)$*-coloring of the processes (where legal means that no two neighbors have the same color).*

Proof: Let us first observe that the processes whose initial color belongs to $1, ..., \Delta + 1$ never modify their color. Let us assume that, up to round r, the processes whose initial colors were in the set $1, ..., r$ have new colors in the set $1, ..., \Delta + 1$ and any two of them which are neighbors have different colors. Thanks to the initial m-coloring, this is initially true (i.e., for the fictitious round $r = \Delta + 1$). Let us assume that the previous assertion is true up to some round $r \geq \Delta + 1$. It follows from the algorithm that, during round $r + 1$, only the processes whose current color is $r + 1$ update it. Moreover, each of them updates it with a color that (a) belongs to the set $1, ..., \Delta + 1$ and (b) is not a color of its neighbors. Consequently, at the end of round $r + 1$, the processes whose initial colors were in the set $1, ..., r + 1$ have new colors in the set $1, ..., \Delta + 1$ and no two of them have the same new color if they are neighbors. It follows that, as claimed, this property constitutes a round invariant from which we conclude that each process has a final color in the set $1, ..., \Delta + 1$ and no two neighbor processes have the same color.

In order to turn this algorithm into a secure one there is one thing that we must concider. Namely, how to choose privately a color different from my neighbours'? So by choosing securely, at the end of the algorithm all nodes are assigned to a colour, without knowing who else has the same one, and what colours its neighbours have.

Algorithm 9 ***Secure Coloring:***

1. *The node who is next in round to update its colour send a notifying message to its neighbours.*

2. *Its neighbours invoke their key-generating algorithm, which outputs a public-private key pair: (p_k, s_k), then all but one of them share their public key with the middle node (i.e. the node who notified them, who is placed in the middle of a star during this procedure). Only the middle node knows which of its neighbour did not send a public key- lets call this neighbour the main neighbour.*

3. *The middle node sends all public keys to its main neighbour.*

4. *The main neighbour encrypts its own public key with the received public keys: $c_i \leftarrow Enc_{p_i}(p_m) \forall i \in \{1, ..., N-1\}$, where p_m is the main neighbour's key, and N is the number of neighbours.*
   Then sends $c_i \forall i$ back to the middle node.

5. *The middle node forwards the messages to its remaining neighbours. (It sends all encrypted values to all neighbours, since it does not know which belongs to which neighbour.)*

6. *These neighbours decrypt the key, then encrypt their colour with it.*
   $p_m = Dec_{s_i} \forall i$
   $c_i = Enc_{p_m}(colour_i)$
   They send back the encrypted colours to the middle, who again forwards it to the main neighbour.

7. *The main neighbour decrypts the colours, then creates polynome with roots of the received colours.*
   $colour_i = Dec_{s_m}(c_i)$
   $p(x) = (x - colour_1) \cdot (x - colour_2) \cdot ... \cdot (x - colour_{n-1}) = a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} \cdot ... \cdot a_0$
   Then it sends back the polynom to the middle node.

8. *The middle node can try any colour on the polynom, will get non-zero for a good colour.*

Note: We need an assymetric key to encrypt the colours, because with a symmetric one the middle node could easily find out the colours and the key as well.

Security in the presence of semi-honest adversaries:

1. *Privacy:* The main node learns the colour of the other neighbours, and moreover the middle node can also learn all possible coulours it can get, thus it learns which are occupied by its neighbours, although it cannot identifiy them.

2. *Independence of inputs:* From the ramining colours the middle node can choose independently from a particular neighbour, but - trivially - not from the set.

3. *Output delivery:* Complies trivially, since the output is not delivered.

4. *Correctness:* Complies.

5. *Fairness:* Complies.

Security in the presence of malicious adversaries:
The above algorithm works only if the nodes are honest-but curious. In case of a malicious adversary, moreover lets say that the middle node is malicious, then it can substitute the main neighbours public key with its own,

thus learns all its neighbours colours. So what can we do in this case? Nothing. This algorithm can easily be attacket with the old man-in-the-middle attack strategy; thus it can be only applied when all parties want to reach the correct final solution.

Complexity:
$N + N - 1 + 1 + 1 + N - 1 + N - 1 + 1 + 1 = 4N + 3$, where N is the number of neighbours.

Chapter 4

Future Challenges

In this thesis we have seen the result from the combination of three fields, distributed programming, secure multi-party computations and privacy preserving data mining. The two latter are closely related, but introducing the first one brought new flavour and interesting new questions to answer in the future. P2P networks only work under well organized and maintained conditions, on heuristical networks, such as a mesh or sensor network we cannot assume its existence. The need for secure communication networks is rising continuously. In the future the newly created algorithms can be used as sub-protocols in more complex algorithms, such as secure distributed max clique, maximum independent edge or vertex computations, Steiner-tree approximations and many more.

Bibliography

[1] Yehuda Lindell, Benny Pinkas, *Secure Multiparty Computation for Privacy-Preserving Data Mining*, The Journal of Privacy and Confidentiality (2009)

[2] Jaideep Vaidya, Chris Clifton, *Privacy Preserving Association Rule Mining in Vertically Partitioned Data*, Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (2002)

[3] Lea Kissner, Dawn Song, *Privacy-Preserving Set Operations*, Advances in Cryptology â CRYPTO (2005)

[4] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, *Introduction to Data Mining*, Pearson Education, Inc. (2006)

[5] Michael Raynal, *Distributed Algorithms for Message-Passing Systems*, Springer-Verlag Berlin Heidelberg (2013)

[6] Jonathan Katz, Moti Yung, *Applied Cryptography and Network Security*, 5th International Conference, ACNS (2007)

[7] Ronald Cramer, *Introduction to Secure Computation*, Springer-Verlag London, UK (1999)

[8] Nancy A. Lynch, *Distributed Algorithms*, The Morgan Kaufmann Series in Data Management Systems (1996)

[9] Yehuda Lindell, Benny Pinkas, *Privacy Preserving Data Mining*, CRYPTO '00 Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology (2000)

[10] Hagit Attiya, Jennifer Welch, *Distributed Computing*, Wiley Series on Parallel and Distributed Computing (2004)

[11] Keith B. Frikken, Mikhail J. Atallah, *Privacy Preserving Route Planning*, Proceeding of the ACM workshop on Privacy in the Electronic (2004)

[12] Justin Brickell, Vitaly Shmatikov, *Privacy-Preserving Graph Algorithms in the Semi-honest Model*, ASIACRYPT (2005)

[13] Kun Liu, Kamalika Das, Tyrone Granison, Hillol Kargupta, *Privacy-Preserving Data Analysis on Graphs and Social Networks*, (2008)

[14] Troy Raeder, Marina Blanton, Nitesh V. Chawla, Keith Frikken, *Privacy-Preserving Network Aggregation*, Springer-Verlag Berlin Heidelberg (2010)

[15] Dana Ron, *Algorithmic and Analysis Techniques in Property Testing*, Foundations and TrendsÂŽ in Theoretical Computer Science 5.2 (2010)

[16] Oded Goldreich, *Introduction to Testing Graph Properties*, Property testing. Springer Berlin Heidelberg (2010)

[17] Srdjan Capkun, Jean-Pierre Hubaux, Markus Jakobsson, *Secure and Privacy-Preserving Communication in Hybrid Ad Hoc Networks*, No. LCA-REPORT-2004-015. (2004)

[18] Andrew C. Yao, *Protocols for Secure Computations*, FOCS. Vol. 82. (1982)

[19] Oded Goldreich, *Secure Multi-Party Computation*, Manuscript. Preliminary version (1998)