Scalable Decentralized Privacy-Preserving Location Proximity

Alexander. M. Bergersen

Department of Computer Science
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Scalable Decentralized Privacy-Preserving Location Proximity

Alexander M. Bergersen

Department of Computer Science
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Scalable Decentralized Privacy-Preserving Location Proximity.
ALEXANDER M. BERGERSEN

Department of Computer Science
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science
Gothenburg, Sweden 2017

Abstract

This report is the report of a Master Thesis carried out at Chalmers University of Technology by me, Alexander Bergersen. The aim of the Master Thesis was to introduce authentication into decentralized privacy-preserving location proximity without adversely hampering the scalability of such protocols further. This was done by creating two solutions that slightly modified the behavior of a location proximity protocol called “InnerCircle”. The first solution, called the Gossiping solution, tries to introduce authentication by having the base network be an unstructured distributed network where each client substitutes as an authentication authority for each of the clients connected to it, viewed as its neighbors. This solution assumes a majority honest network where each vote of authenticity from each neighbor carries equal weight. The second solution develops the first one by going in a different direction and uses an Identity Based Encryption system with distributed key generation as the basis. Instead of proving authentication it hides information from those not authorized to see it. However, even though the second solution is based on the first one, they are different enough that instead of the testing each solution and showing the progress, the solutions are compared as to which one is better and most fulfill the aim of not adversely affecting the performance of the underlying protocol, InnerCircle, the best. The test results are presented to the reader for each of three cases, baseline, first solution, second solution, in tables and the data is analyzed. This analysis showed that the Gossiping solution without the Identity Based Encryption will eventually always outperform the second solution, even if the Identity Based solution was developed with the goal of performing better than the Gossiping solution.

Keywords: Location Proximity, InnerCircle, Decentralized Cryptography, Distributed Systems

Table of Contents

1.Introduction

Location based services are becoming more and more common in modern everyday life. Over 2500 companies are active location based services companies according to one online resource [2], at the time of writing. The uses of location based services range from allowing shipping companies track their goods to helping users find nearby restaurants and everything in between [1][4]. However, there is a major challenge in location based services in trying to provide privacy to the users without hurting the utility of the services.

There exists a great number of location based services and they are growing in popularity thanks to the abundance of mobile devices [5]. The location based services provide everything from allowing shipping companies to track their wares to allowing individual users the ability to track their phone [1][4]. However, these types of services, especially those designed for individual users, have been used to create other types of applications that could perhaps be considered as privacy intrusive [1][3].

This thesis is centered on location proximity. Where privacy concerned users are willing to reveal if they are within a certain distance of each other, without revealing any other location information. The focus of the thesis is on scalable decentralized privacy-preserving location proximity. The decentralized aspect is to remove the need to trust a third-party with any location information, something which is the current practice today [1]. As location proximity usually involve using homomorphic encryption to preserve privacy, scalability becomes an issue when the decentralized approach is take. This is due to how the number of encryption keys needs to be known scales with the number of users. Therefore, the aim of this thesis is to utilized the decentralized aspect to create a key sharing and authentication protocol that in combination with a decentralized privacy-preserving location proximity protocol, like InnerCircle [1], to provide a scalable solution.

By utilizing location proximity, the privacy of users of location based services can be protected without rendering the services useless [1][5]. By using location proximity both the features of which two users want to find out if they are within a certain distance of each other and a single user wanting to find out if other users are within a certain distance, called mutual location proximity and one-way location proximity, are possible [1].

There exist protocols such as InnerCircle [1] that uses secure multi-party computation to achieve privacy-preserving location proximity. InnerCircle is also decentralized, something that the current solutions for mutual and one-way location proximity are not [1]. It is decentralized to avoid having to trust a third party.

1.1 Structure

The thesis is structured as follows. We start by covering the background theory needed to understand the rest of the thesis. This will be done through a Theory Chapter that introduces all the concepts used in the later chapters and the theory behind them with examples throughout the chapter to illustrate how the different concepts work. The Theory Chapter covers subjects ranging from Distributed Systems to Elliptic Curve Cryptosystems and Location Proximity. After the Theory Chapter comes the Challenges and Approach chapter that introduces the research question and provides the basis for this same research question. The Challenges and Approach chapter also provides the reader with the method used to tackle said research question and provides insight into how the research method was executed. Following this Challenges and Approach chapter comes the solution chapter, in which the two solutions to the authentication problem, that this thesis provides are fully explained. The Solution Chapter also provides a possible implementation for each solution and covers the security considerations for each solution as well. The fifth chapter of this thesis is the Testing and Results Chapter which explains how the testing and data gathering was carried out and presents the results of said testing and data gathering before providing a short conclusion that is established from the same data. The main research question is also answered in the conclusion. After the Testing and Results Chapter comes the final two chapters of this thesis, with the second to last one covering the possible Social or Ethical consequences that fully practical location proximity could have and the final chapter providing a starting point for possible Future work that can build and improve upon this thesis.

2. Theory

This chapter provides all the theory and concepts needed to understand where the authentication issue, more on that in the next chapter, this thesis aims to solve comes from and to understand the presentation of the solutions in the Solutions Chapter. The aim is to provide an introduction to every concept this thesis is built upon without going to much into detail and to utilize examples to help with understanding the concepts.

The reader can safely skip this chapter if they feel they have a sufficient grasp of the following concepts: Distributed Systems, Public-key Cryptography, Location Proximity, Secure Multiparty Computation, InnerCircle, Elliptic Curve Cryptography, Identity Based Encryption and Distributed Identity Based Encryption.

2.1 Distributed Systems

Distributed systems, or distributed computing, is the art of pooling together resources from nodes connected together in some type of network [11]. These pooled resources can then be accessed and used by any node in the system [11]. This can be utilized in many different instances such as needing more computing power to calculate some algorithms than is readily available on a single node. This node can then also “loan” resources from other nodes that they have put in the pool and thus increase its own effective computing power. There are also instances where this pool is a type of extra storage, i.e. every node provides some of its storage to be accessed and used by other nodes in this system. In essence this type of distributed computing and distributed storage is what most recognize as “Cloud computing” and “Cloud storage”. Figure 1 and figure 2 below examples of how cloud storage is viewed and how it could be built.

Cloud storage
User

Figure 1. How the cloud storage looks from the viewpoint of the end user.

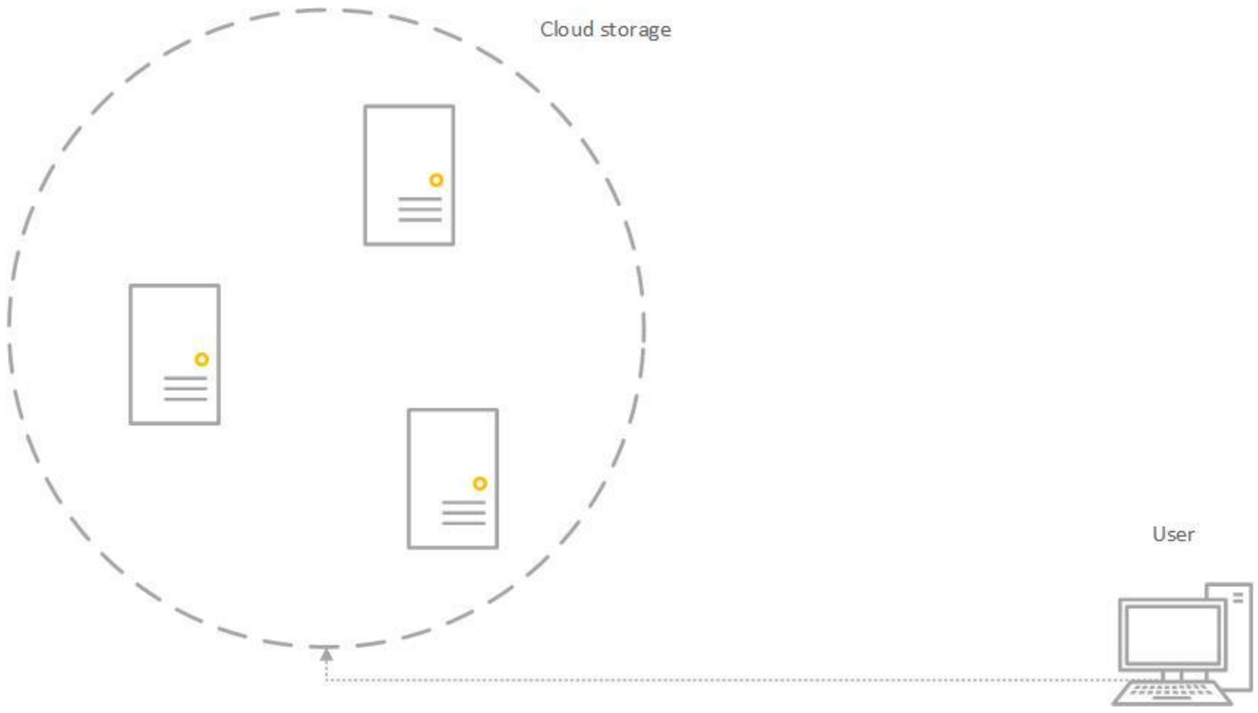


Figure 2. How the cloud storage is actually built.

There are a few different ways the network and resource division can be set-up, such as a server keeping track of all resources available on all clients in the network and handling the resource requests in the network. However, we will focus on Peer-to-Peer(P2P) networks in this thesis. In P2P networks all nodes are equal, there is no typical server-client architecture, instead all nodes work both as servers and clients at the same time [12]. What this means is that essentially every single node in a P2P network has to be connected to *at least one other node* in the network.

P2P networks are divided up into two categories, *structured networks* and *unstructured networks* [12]. Structured networks are constructed using some deterministic method, most commonly a *distributed hash table*(DHT) is used [12]. In these DHT systems all items are assigned a *key* and each node in the network is assigned some *random* number [12]. In this system each *key* is deterministically uniquely mapped to the identification number of some node. Unstructured networks on the other hand are constructed using randomized algorithms with the idea of each node *n* maintaining some list *L* of neighbors [12]. The neighbors in the list *L* can be as considered all the nodes the node *n* is connected to and is constructed in some random manner, often by simply picking random, hopefully live, nodes in the network. Data items are then randomly divided up among the nodes. There also exist some *hybrid networks* that are a mix between structured and unstructured networks, utilizing different ideas from those two categories to create their network topology and divide up the data items between the different nodes.

However, it is not only the construction of network topology and data item division that differs between structured and unstructured networks, they also, quite naturally, differ in how they conduct searches for specific data items in the network [12]. A structured network has quite efficient search since it can utilize a hash table to keep track of the (key, number) pair stored in the DTH to know which data item is stored on which node. A search of a certain data number then simply consists of searching the hash table for the key, getting the number in response and setting up a connection with the corresponding node. In an unstructured network on the other hand, a node needs to flood the network with a search query [12], often this is done by first querying in the node's neighbors for the item and the neighbors either query the nodes in their neighbors list, if it does not contain the item, or respond with the item to the node that queried originally. This type of “flooding” leads to a very high amount of traffic in the network.

An interesting type of distributed network is a network setup called *Content Addressable Network*, or CAN for short, it will be briefly explained below. Part of the suggested implementations in the solution chapter is based on the CAN distributed network.

2.1.1 Content Addressable Network (CAN)

A *Content Addressable Network* (CAN) is a DHT based structured network that uses a *k-dimensional Cartesian coordinate space* that it divides up among all nodes to assign data items to the different nodes [12] [13]. What this means is that each node *n* participating in the system is assigned a region in the coordinate space that the node is responsible for. Each time a data item is added to the system it is assigned a random unique point in the coordinate space and whichever node is responsible for the region in which this point lies becomes responsible for the data item [12] [13].

When a node *X* wants to join the system, it picks a random point in the coordinate space and looks up which node *Y* is responsible for the region in which that points fall. *X* then continues by contacting the node *Y*, which then proceeds by splitting its region in two parts and hands over the responsibility of one of the halves to *X*. Whenever *X* decides to leave the system, the region it was responsible for is assigned to one of the nodes responsible for any of the neighboring regions. Since this can lead to asymmetric partitioning of the coordinate space, a background process is started that repairs the entire coordinate space.

Following, figure 3 to 5 is an illustrative example of CAN using a 2-dimensional coordinate space.

Figure 3. Step 1. Initially there is only a single node in the system responsible for the entire coordinate space.

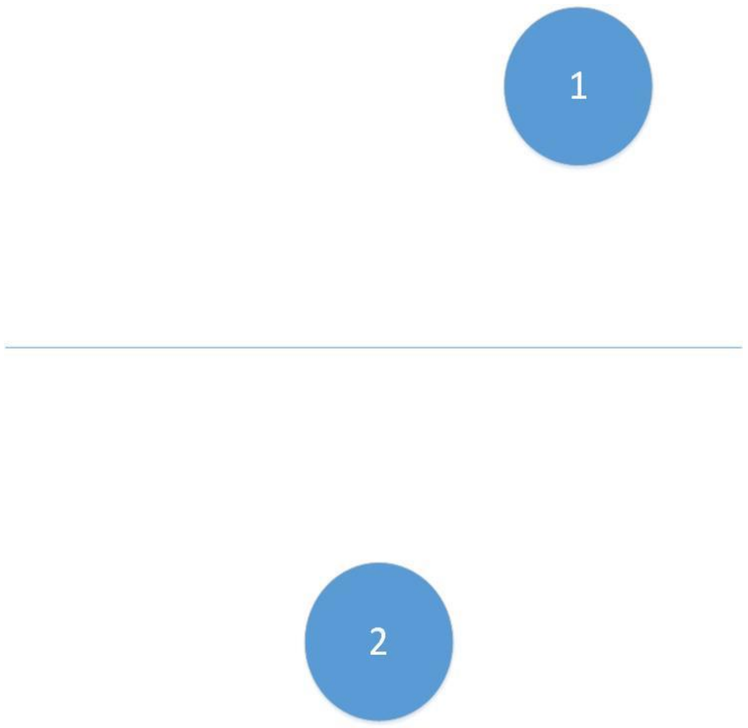


Figure 4. Step 2. A second node (2) joins the system and the first node (1) splits the coordinate space with the new node along the y-axis. The two nodes are now responsible for half each.

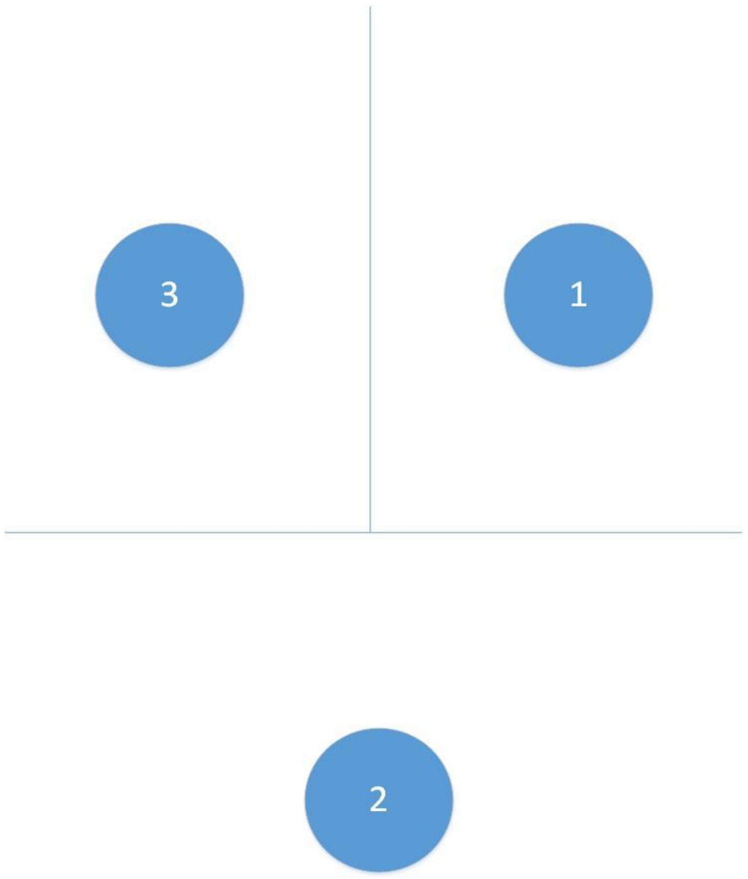


Figure 5. Step 3. A third node (3) joins and is directed to the node (1). (1) splits the coordinate space it is responsible for once more, now along the x-axis instead and hands over the responsibility for one of the halves to (3).

2.2 Cryptography

This section will provide a background on cryptography in general and more specific the different types of cryptographic concepts that will be employed in the cryptographic solution and to some extent in the distributed solution.

There are two main fields of cryptography, symmetric cryptography and asymmetric cryptography, these fields are also known as shared-key cryptography and public-key cryptography respectively [14]. While this thesis will focus on cryptographic systems found in the public-key cryptographic field, the following segments will provide a short explanation of shared-key cryptography and the differences between the two fields.

The biggest difference between public-key cryptography and shared-key cryptography can be derived from the names of the two fields. In shared-key cryptography both the sender and the receiver need to know some shared key that is used for both decryption and encryption while in public-key cryptography the sender uses the receiver's public-key to encrypt the message and the recipient use their private key to decrypt said message [14]. A more in-depth explanation of public-key cryptography will be provided in a later section.

Shared-key cryptography was the original, and for a long time the only, type of encryption that existed. The fact that both the sender and receiver need to have shared knowledge of the key **K** being used makes key distribution the main problem of shared-key cryptography. Solving this key distribution problem was one of the main motivations behind public-key cryptography.

2.2.1 Public-key Cryptography

Public-key cryptography was first introduced by Diffie and Hellman in a 1976 paper called "New Directions In Cryptography" [15]. In said paper Diffie and Hellman proposed the now widely used concept of each user providing a type of encryption procedure E to the general public, whilst at the same time keeping secret the details of the corresponding decryption procedure D [15]. For a public-key cryptosystem to be valid the following four properties must hold for the encryption and decryption procedures [15]:

1. Decrypting the encrypted version of a message M gives M, D(E(M)) = M.
2. E and D are easy to compute when some secret "trapdoor" information is known.
3. The user does not provide an easy way to compute D by revealing E to the public. This means that there is no easy way to compute the secret procedure D using information gained from the public procedure E. Only the user can decrypt the messages encrypted with E or, since he has the "trapdoor" information, easily compute D.
4. M is the result of first decrypting some message M and then encrypting it, E(D(M)) = M.

Functions that satisfy the above four properties are known as "trapdoor one-way permutations" [16]. The functions are "one-way" because they are easy to compute in one direction but difficult, or at least take a lot of effort, to compute in the other direction [16]. The "trapdoor" part of the name is due to the fact that the reverse direction become easier to compute once some type of private "trapdoor" information is learned [16]. It is by the satisfaction of the fourth property a "trapdoor one-way function" becomes a permutation.

Following, figures 6 and 7, is an example of how Bob would send a private message to Alice using a Public-key cryptosystem

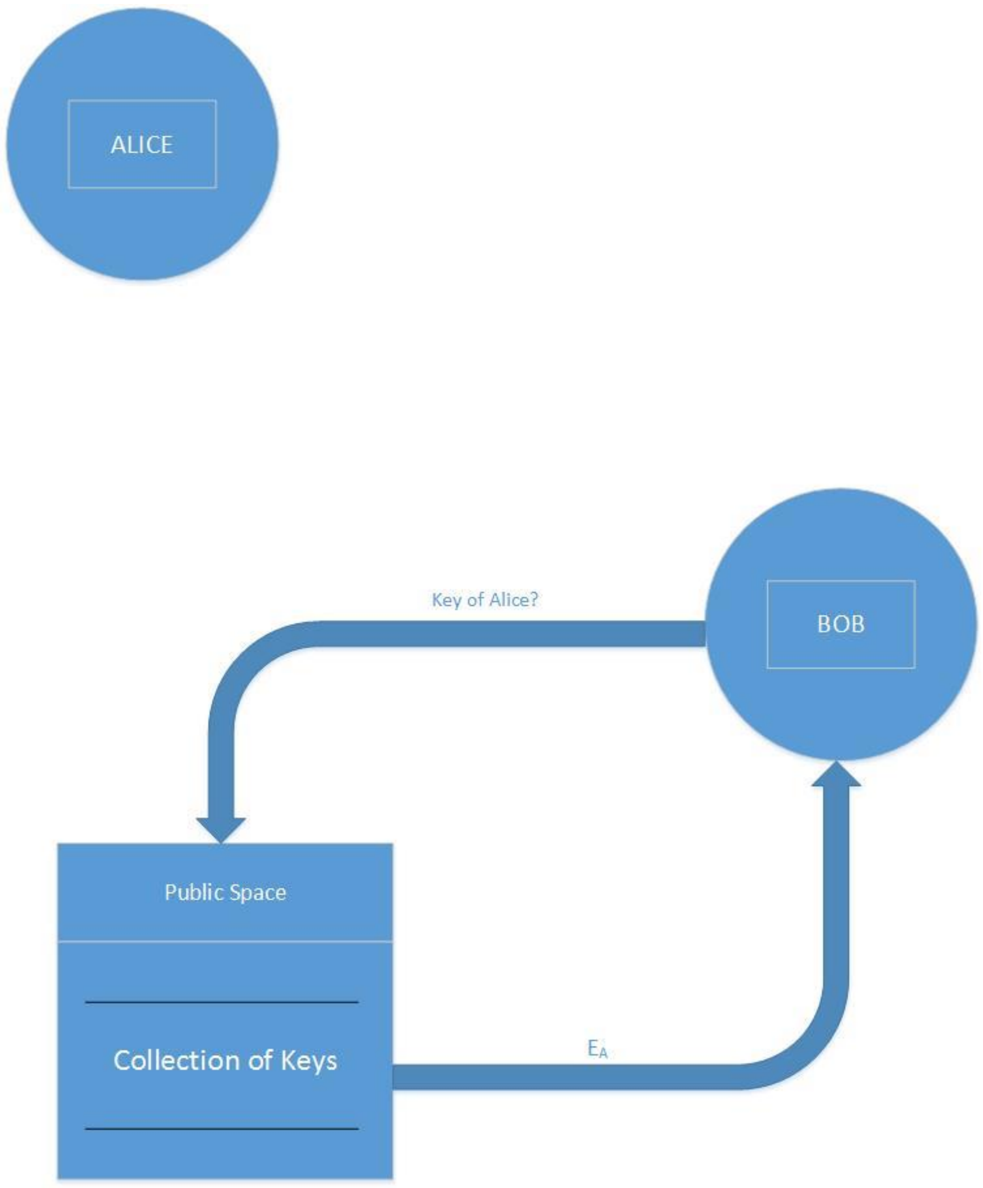


Figure 6. Step 1. Bob starts with retrieving Alice's encryption key, E_A, from some public place.

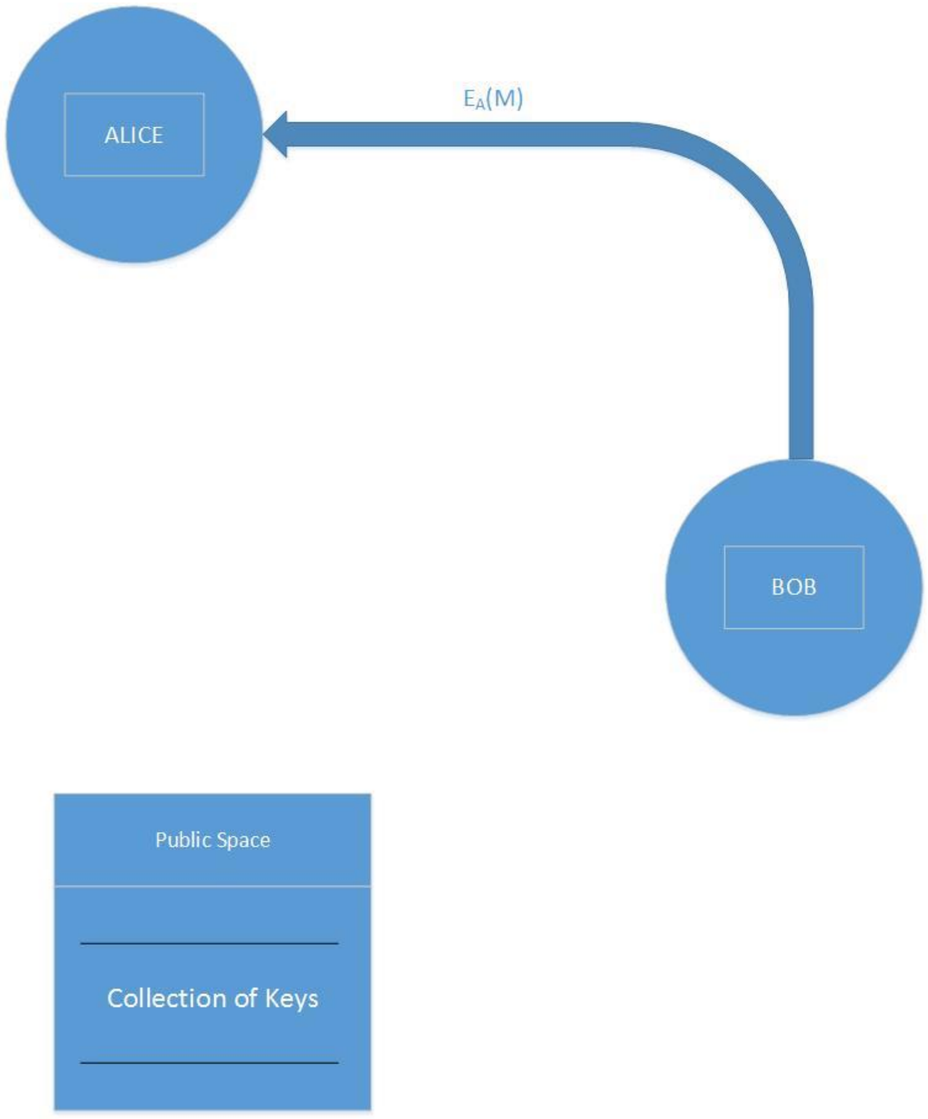


Figure 7. Step 2. Bob now proceeds with sending Alice the encrypted message $E_A(M)$. After Alice receives $E_A(M)$, she decrypts it using her decryption key, D_A, by computing $D_A(E_A(M)) = M$. Only she can decrypt $E_A(M)$ thanks to property (3) of the cryptosystem. If Alice wants to respond to Bob, she can encrypt a response by fetching E_B from the same public space.

Instead of fetching the encryption keys from the public space as shown in figure 7 above, Alice and Bob could exchange keys with each by sending each other their public key in an unencrypted message [16]. This is how keys are exchanged in a decentralized environment and leads to problems regarding authentication, which will be covered in detail later.

The first practical public-key cryptosystem to be put forward was the RSA cryptosystem, which was provided in a paper by Rivest, Shamir and Adleman in the 1977 called “A method for obtaining digital signatures and public-key cryptosystems” [16]. In said paper they proposed a public-key cryptosystem based on the assumption that the factorization of semi prime number into its prime factors is a hard to solve problem, i.e. that there is no polynomial time algorithm that return the primes p and q when provided the with semi prime $n = p * q$ [16].

2.2.2 Location Proximity

This section will provide an explanation of what Location Proximity means and how it works. As well as providing an introduction to Secure Multiparty Computation and the protocol that provides the basis for this thesis, InnerCircle, both of which will have their own sections, 2.2.2 and 2.2.3 respectivly, with more detailed explanations later in the thesis.

Location Proximity aims to solve the Location Privacy Challenge, i.e. the challenge of providing privacy to the end users of Location Based Services whilst still providing the benefits and utility of such services [1][5]. In Location Proximity the focus is on the case where two users, such as Alice and Bob, want to know if they are within some distance r from each other, whilst not revealing any other information about their locations [1]. This means that Location Proximity is not naturally applicable to the current Augmented Reality Game, ARM, Pokémon GO that is, as of writing, taking the world by storm, since it utilizes the GPS information gained from the user's devices to keep track of their location at all time to function as intended.

There are two main types of location proximity, mutual location proximity and one-way location proximity [1]. Mutual location proximity is where both participants are informed of the results and is useful in situations like collision detection [1]. One-way location proximity on the other hand is when only one of the participants learns of the results [1]. This can be used for instance when a user wants to find out if there is a person with a certain type of profession nearby. In the case of one-way location proximity one of the participants can be considered active, the participant that is requesting the information, and one is inactive, the participant registered with the desired type of profession. The result of the request in this example is only of interest to the active participant.

2.2.2.1 Current common practice

Currently the common practice is to have a trusted third party that the users send their location data to in the mentioned scenarios [1]. The third party then perform the necessary calculations and inform either both parties or just the active one, depending on which type of location proximity is in play, of the results. However, this practice is not applicable to this thesis, as both the thesis itself and the work it is based on, aim to provide a decentralized solution. This is to avoid having to trust a third party [1].

The solution presented in the InnerCircle paper [1] suggests utilizing *Secure Multiparty Computation* to provide a decentralized location privacy solution in the location proximity settings mentioned above.

2.2.3 Secure Multiparty Computation

A secure multiparty computation protocol, SMC protocol, is a type of protocol that allows k participants to collectively calculate some result r by all of the participants contributing some private value x_k, without revealing x_k to the other participants [17]. The SMC protocols achieve this by encrypting each value x_k using a public-key encryption scheme that has a *homomorphic* attribute, i.e., a *homomorphic encryption scheme* [14]. This allows users to calculate some function of a plaintext, only needing the knowledge of the ciphertexts of the other users' values and the public-key. However, for the result to be correct, all values must have been encrypted using the same public-key. This means that the entity that created the public-key and private-key combo can decrypt all the ciphertexts c_k corresponding to the value x_k.

2.2.3.1 Homomorphic Encryption Scheme

Homomorphic encryption schemes exist in two different forms, partially homomorphic or fully homomorphic [14]. The difference between them is that a partially homomorphic encryption scheme supports either addition of two ciphertexts or the multiplication of two ciphertexts, i.e., it is either additively homomorphic or multiplicative homomorphic [14], while a fully homomorphic encryption scheme supports both [14].

Encryption protocols such as InnerCircle needs something called additively homomorphic encryption [1]. For an encryption scheme to be additively homomorphic there must exist some addition function $\oplus$ such that $E(m_1) \oplus E(m_2) = E(m_1 + m_2)$ is true. Both sides of that equation will hold true if the evaluation of the following, $D(E(m_1) \oplus E(m_2)) = D(E(m_1 + m_2))$, is true.That both $D(E(m_1) \oplus E(m_2))$ and $D(E(m_1 + m_2))$ evaluates to $m_1 + m_2$.

The following example will demonstrate how a SMC protocol using an additively homomorphic encryption schemes calculates the sum of Alice, Bob and Eve's private values, x_A, x_B and x_E respectively, without leaking said values to the other participants.

Example 1

Alice, Bob and Eve all possess some secret value, x_A, x_B and x_E respectively, and want to find out the sum of the values, but they don't want to reveal these values to each other. They decide on the following scheme:

All three send their secret value to the other two encrypted using *their own public key*, k_A, k_B and k_E respectively. That is, Alice sends $Ek_A(x_A)$ to Bob and Eve, Bob sends $Ek_B(x_B)$ to Alice and Eve, and Even sends $Ek_E(x_E)$ to Alice and Bob, where $Ek_y(x_y)$ stands for y's secret value encrypted with the public key belonging to y.

With these values Alice can now calculate $Ek_E(x_E) \oplus Ek_E(x_A) = Ek_E(x_E + x_A) = Ek_E(x_{EA})$ and send$Ek_E(x_{EA})$ back to Eve, Eve can now decrypt $Ek_E(x_{EA})$ to $x_{EA}$and do the same with the values received from Bob and the perform the the following calculation $x_{EA} + x_{EB} - x_E = x_{EAB}$ to get the sum of the three values x_{EAB}. The same procedure will be repeated with values encrypted using Alice's and Bob's public-keys, granting all three information of the total sum.

The above example is not perfect, Eve can very easily calculate $x_A$and x_B, from $x_{EA}$and x_{EB}respectively, but it should provide an overview for how SMC could work in practice.

2.2.4 InnerCircle

InnerCircle was proposed in a paper called "Innercircle: A parallelizable decentralized privacy-preserving location proximity protocol" [1] as a possible solution to the Location Privacy Problem and is a *decentralized privacy-preserving location proximity protocol* [1]. The privacy-preserving location proximity part means that InnerCircle is a protocol designed to allow two users, Alice and Bob for example, to find out if they are within a certain distance of each other without revealing any other information, such as their locations [1]. While the decentralized aspect says that InnerCircle achieves this without having to rely on a trusted third-party.

Part of the solution for achieving decentralized privacy-preserving location proximity in the InnerCircle protocol involves encrypting the private location values using an additively homomorphic encryption scheme, with the encryption scheme supporting the following operations:

$$E(m_1) \oplus E(m_2) = E(m_1 + m_2)$$

$$\neg E(m_1) = E(-m_1)$$

$$E(m_1) \odot m_2 = E(m_1 \cdot m_2)$$

Where $\oplus$ is an addition function, $\odot$ is a multiplication function and $\neg$ is the negation function [1].

There must also be some randomization function R such that $R(E(m_1)) = E(0)$if $m_1 = 0$, and $R(E(m_1)) = E(L)$ otherwise, where L is a random value not equal to 0 [1].

The location values of the active party, the one that instigates the protocol wanting to know if the passive party is within some distance *r*, encrypts its location values using its own public-key *k* and then sends them to the passive party together with the distance *r* [1]. To perform the necessary calculations, to decide if the passive and active party are within distance *r* from each other, the passive party also need encrypts their location values under the active party's public-key, *k* [1]. The passive party then uses the operations shown above to calculate the distance, also encrypted under *k*, and performs a last check to see if they are within *r* of each other before sending back a response encrypted using *k* that the instigator then decrypts using its private-key *K* [1]. Figure 8 illustrates the protocol.

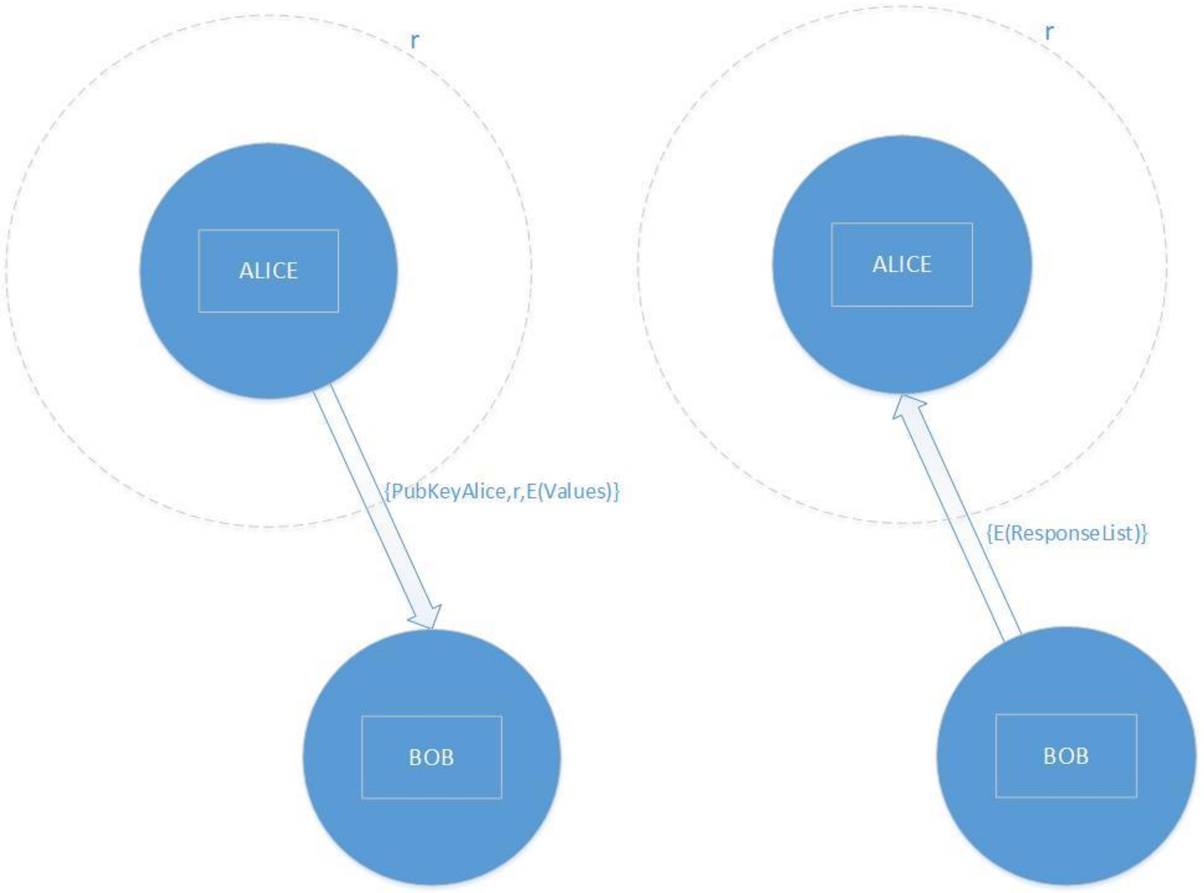


Figure 8. Alice and Bob implement the InnerCircle protocol for location proximity. (a) Alice sends a location proximity query to Bob containing all the needed data. (b) Bob responds with the list of the encrypted values, written as E(ResponseList). that may or may not contain an encrypted 0. In this picture the list will not contain said 0 value.

InnerCircle uses the Euclidean distance calculation, $\sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2} = d$, to calculate the distance between the participants [1]. However, since the encryption scheme only supports a few operations, the once mentioned earlier, this equation must be rewritten to $(x_1^2 + x_2^2 + 2x_1x_2) - (y_1^2 + y_2^2 + 2y_1y_2) = d^2$[1]. The protocol on the passive participant then subtracts every value between 0 and $r^2$from the squared distance, d^2. The result is then stored in list either as a random encrypted number, if $r^2 - d^2 \neq 0$ or as 0 encrypted using *k* through the function R [1]. This list is then shuffled and returned to the instigator, who then decrypts all the values and checks if the list contains the value 0. If that is the case, the participants are within distance *r* from each other [1].

Example 2

Alice has the position $(x_A, y_A) = (5, 3)$ and Bob has the position $(x_B, y_B) = (8, 6)$. Alice wants to know if Bob is within a distance 10 of her, $r = 10$. She starts of with encrypting the values of x_A, y_A, x_A^2, y_A^2 with her public-key *k* and sends the following message to Bob: $[E_k(x_A), E_k(y_A), E_k(x_A^2), E_k(y_A^2), r]$ (Note that all encryptions written $E_k()$in this example are encrypted using Alice's public-key).

When Bob receives this message, he proceeds to use the rewritten Euclidean distance equation to calculate $E_k(d^2)$.

$$(E_k(x_A^2) \oplus E_k(x_B^2) \oplus ((E_k(x_A) \odot x_B) \odot 2)) \oplus (\neg (E_k(y_A^2) \oplus E_k(y_B^2) \oplus ((E_k(y_A) \odot y_B) \odot 2))) = E_k(d^2)$$

$$(E_k(25) \oplus E_k(64) \oplus ((E_k(5) \odot 8) \odot 2)) \oplus (\neg (E_k(9) \oplus E_k(36) \oplus ((E_k(3) \odot 6) \odot 2))) = E_k(d^2)$$

$$(E_k(25 + 64) \oplus (E_k(5 \cdot 8) \odot 2)) \oplus (\neg (E_k(9 + 36) \oplus ((E_k(3 \cdot 6) \odot 2))) = E_k(d^2)$$

$$(E_k(89) \oplus (E_k(40) \odot 2)) \oplus (\neg (E_k(45) \oplus ((E_k(18) \odot 2))) = E_k(d^2)$$

$$(E_k(89) \oplus E_k(40 \cdot 2)) \oplus (\neg (E_k(45) \oplus E_k(18 \cdot 2)) = E_k(d^2)$$

$$(E_k(89) \oplus E_k(40 \cdot 2)) \oplus (\neg (E_k(45) \oplus E_k(18 \cdot 2)) = E_k(d^2)$$

$$(E_k(89) \oplus E_k(80)) \oplus (\neg (E_k(45) \oplus E_k(36)) = E_k(d^2)$$

$$E_k(89 + 80) \oplus (\neg E_k(45 + 36)) = E_k(d^2)$$

$$E_k(169) \oplus (\neg E_k(81)) = E_k(d^2)$$

$$E_k(169) \oplus E_k(-81) = E_k(d^2)$$

$$E_k(169 - 81) = E_k(d^2)$$

$$E_k(88) = E_k(d^2)$$

After that calculation is complete, Bob takes every single value from 0 to 100, i.e. from 0 to r^2, and puts the result of $R(E_k(88) \oplus (\neg (E_k(n))$, into a list *L*, where *n* is the current number between 0 and 100 and *R* is the randomization function described earlier. When this is complete Bob sends the list *L* back to Alice and Alice proceeds with decrypting all the values $E_k(i)$in the list *L*. Once the decryption process is complete, Alice checks the list of the now decrypted values for a 0 and since 88 < 100, she finds a 0 in the list and thus knows that she and Bob are within a distance of 10 from each other.

2.3 Elliptic Curves

An elliptic curve is the set of points that solve some equation of the form $y^2 = x^3 + ax + b$ and the graph of these points possesses neither cusps nor does it intersect itself [18], see figure 9 for an illustration. Cusps being a point(s) on a curve which is not differentiable, i.e. there exists no tangent for the point(s). The point (0,0) on the curve $y^2 = x^3$ is an example of a cusp.

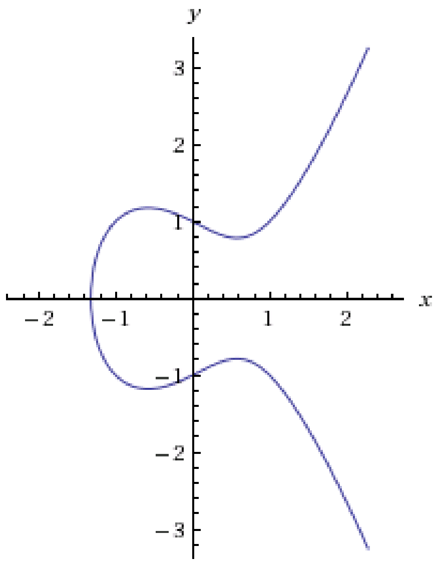


Figure 9. This is the graph of the elliptic curve $y^2 = x^3 - x + 1$[7].

Elliptic curves have two attributes that are very important for them as mathematical concepts. Firstly, they are symmetric along the x-axis, i.e. the part below the x-axis, the negative part of the y-axis, mirrors the part above the x-axis, the positive part of the y-axis [6] [18]. Secondly, if a straight line is drawn through an elliptic curve, the straight line will intersect the curve in no more than three points [6] [18]. These two attributes are used to define addition for points on elliptic curves as follows [6] [18], if we have two points on a curve E, P and Q, and we draw a line L through these two points, that line will intersect in a third point on E, denoted -R. Since the curve E is symmetric about the x-axis, as stated previously, we can flip -R over the x-axis to its mirror image, R, thus P + Q = R. In the case of P + P, the line L being drawn is the tangent for the curve E in the point P and R being the point intersected by this tangent on E. In the case P + P, L is counted as intersecting E twice in the point P, thus P + P = 2P. This fact is also the basis for multiplication of a point, as it is simply treated as repeated addition, e.g. 3P = 2P + P = R + P = S. Figure 10 illustrates how the addition on elliptic curves is performed.

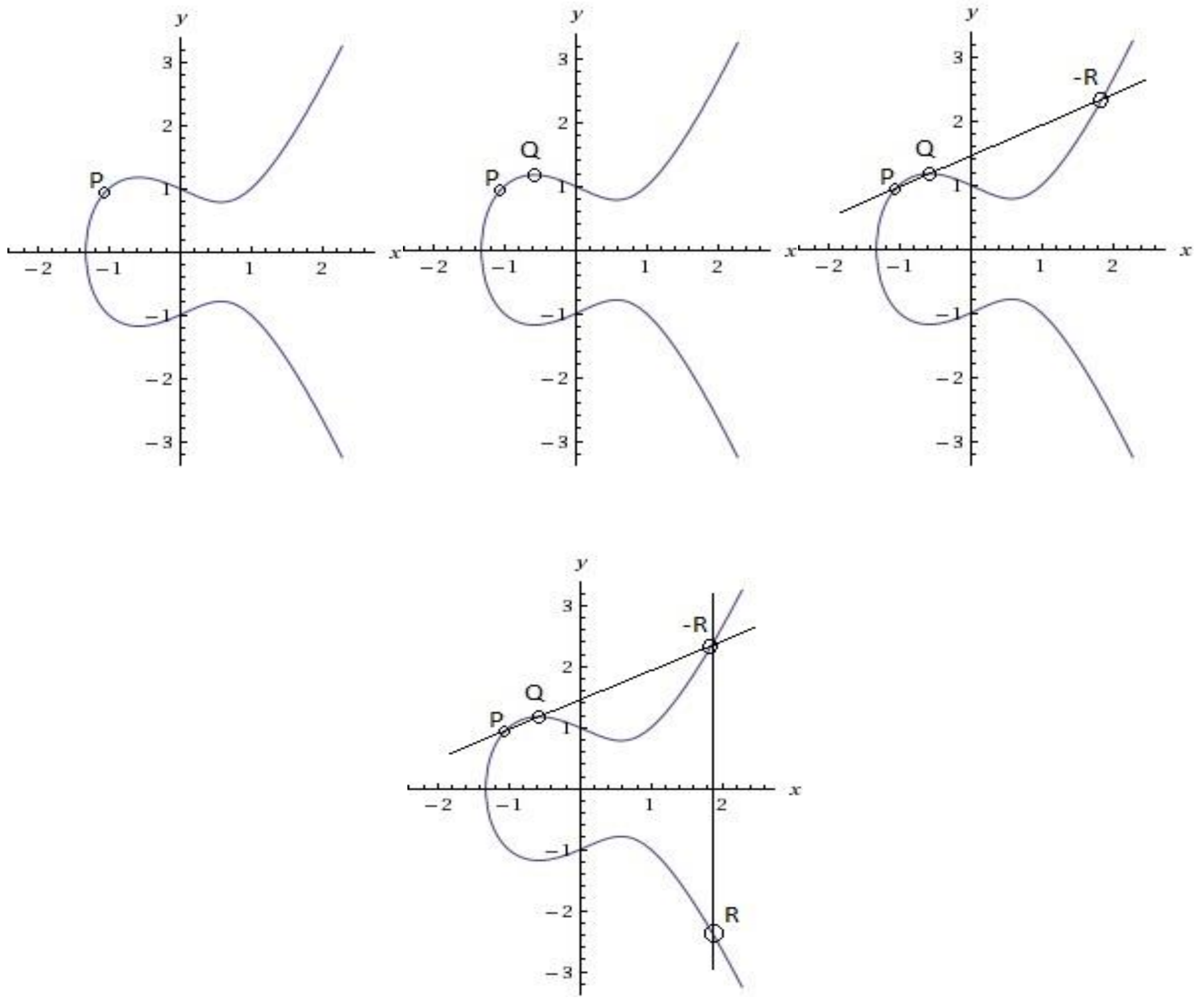


Figure 10. The above pictures show point addition on the curve $y^2 = x^3 - x + 1$

The type of elliptic curves we are interested in are elliptic curves over finite fields, prime fields to be exact. Where a prime field F for prime *p* is the resulting set of the function f: x mod p. These types of curves are generally written as the curve E over the field F_p or E/F_p [6] [18]. E/F_p is the subgroup of the curve E, the set of all points in E on the field F_p, and in these types of curves **a** and **b** are points on the field F_p [6] [18].

There are also a few important terms in relation to elliptic curves that needs to be understood for the following sections and they are the following:

Generator g of a cyclic group G: The generator g is an element in G which through the use of an associative function, or the functions inverse, can create every single other element in G.

Order of a group G (such as E/F_p): The order of a group is the number of elements in the group.

Cofactor: The relation between the order of a group G and the order of one of its subgroups. A group G can thus have many different cofactors.

2.4 Elliptic Curve Cryptography(ECC)

Elliptic Curve Cryptography, ECC, is a type of public-key cryptography and its security is based on the assumption that the “Elliptic Curve Discrete Logarithm Problem”, abbreviated to ECDLP, is hard to solve, i.e. that there is no polynomial time algorithm that solves it [18]. The ECDLP is that given two points, P and Q, and some cyclic group G, in this case an elliptic curve, where nP = Q in G, there is no algorithm that can compute n in polynomial time [18]. This leads to nP being a one-way trapdoor function, which is easy to compute one way, but difficult to reverse, as explained in the Public-key Cryptography section, and, as also explained in the Public-key Cryptography section, one-way trapdoor functions are the most essential part of any Public-key Cryptography system.

The main advantage of ECC over other public-key cryptosystems, such as RSA, is the smaller key size needed to obtain the same bit security [20]. For example, to achieve 128-bit security RSA requires a key size 3072 bits while ECC only requires the key size to be 256 bits to achieve the same level of security [20].

To be able to utilize ECC the participants need to agree on a few so called domain parameters, these domain parameters are the defining elements of the curve E over the field F. In the case of using curves over prime fields these parameters are:

p, the defining prime for the prime field F_p.

The constants **a** and **b,** used in the defining equation for the curve E.

The generator **g** for the cyclic subgroup G.

The order of the subgroup G, denoted **n**.

And finally, the cofactor, **h**, between G and E/F_p.

The domain parameters are thus (**p**, **a**, **b**, **g**, **n**, **h**) when using elliptic curves defined over prime fields.

Following will be an example of Elliptic Curve Diffie-Hellman to illustrate how ECC works using Alice, Bob and Eve as the participants with Eve working as a man in the middle intercepting Alice’s and Bobs messages to each other.

Example 3

Assume that the domain parameters (**p**, **a**, **b**, **g**, **n**, **h**) have already been established, then Alice and Bob start by generating their own private and public keys. Alice generate her private key $\mathbf{r_A}$, which is some random number generated by Alice, and calculates her public key as $\mathbf{x_A} = \mathbf{r_A g}$. Remember that g is the generator for the subgroup G of the curve E/F_p. Bob does the same to produce $\mathbf{r_B}$ and $\mathbf{x_B} = \mathbf{r_B g}$. Alice and Bob proceeds by exchanging their public keys, $\mathbf{x_A}$ and $\mathbf{x_B}$, over some insecure channel where Eve, in her capacity as a man in the middle, can intercept their messages. Alice and Bob can now calculate a shared secret $\mathbf{s} = \mathbf{r_A x_B}$ and $\mathbf{s} = \mathbf{r_B x_A}$. The shared secret **s** is the same for Alice and Bob, since $\mathbf{s} = \mathbf{r_A x_B} = \mathbf{r_A}(\mathbf{r_B g}) = \mathbf{r_B}(\mathbf{r_A g}) = \mathbf{r_B x_A}$, and can be used by Alice and Bob as the shared secret in Symmetric Encryption. However, Eve cannot calculate **s** even if she has intercepted both $\mathbf{x_A}$ and $\mathbf{x_B}$, since she does not have access to neither $\mathbf{r_A}$ or $\mathbf{r_B}$ even if she has $\mathbf{x_A}$ and $\mathbf{x_B}$. This is because, as previously established, solving either $\mathbf{r_A}$ or $\mathbf{r_B}$ from $\mathbf{x_A}$ or $\mathbf{x_B}$ respectively would require Eve to solve the Elliptic Curve Discrete Logarithm problem.

2.4.1 Elliptic Curve ElGamal

One of the encryption schemes that fulfill the criteria of being additively homomorphic [14], and the one this author has chosen to use for the InnerCircle protocol, is called the ElGamal encryption scheme which was described by Taher Elgamal in 1984 [21]. The Elgamal encryption scheme is based on the Diffie-Hellman key exchange protocol, of which the Elliptic Curve version was shown in example 3 in section 2.4, and, just as the non-Elliptic Curve version of the Diffie-Hellman protocol, it too is based on the discrete logarithm problem explained earlier in section 2.4 [21]. Also, just as the Diffie-Hellman protocol, it has an Elliptic Curve version that will be explained below in example 4 [22].

Example 3

Assume that, just as in example 3, the domain parameters (**p**, **a**, **b**, **g**, **n**, **h**) have been agreed upon by Alice and Bob. Alice begins by choosing some random **x** within the range of 0 and p and then proceeds by calculating $\mathbf{k_A} = \mathbf{gx}$. Alice publishes $\mathbf{k_A}$ as her public key and keeps **x** secret as her private key. To send a private message **m** to Alice, Bob starts by choosing some random **y** in the range of 0 and p and calculates **r** = **gy**. Bob then proceeds by mapping the message **m** onto an element in the subgroup G of the curve E/F_p, **m***. Finally, Bob calculates the ciphertext as $(\mathbf{c_1}, \mathbf{c_2}) = (\mathbf{r}, \mathbf{m^*} + \mathbf{k_A y})$.

To decipher the ciphertext $(\mathbf{c_1}, \mathbf{c_2})$, Alice calculates $\mathbf{m^*} = \mathbf{c_2} + (-(\mathbf{c_1 x}))$. This is correct since

$\mathbf{c_2} + (-(\mathbf{c_1 x})) = (\mathbf{m^*} + \mathbf{k_A y}) + (-(\mathbf{rx})) = (\mathbf{m^*} + \mathbf{gxy}) + (-(\mathbf{gxy})) = \mathbf{m^*} + \mathbf{gxy} - \mathbf{gxy} = \mathbf{m^*}$ [14]. Alice now only needs to reverse the mapping of **m*** to receive the original message **m**. Note that just as in the Diffie-Hellman example any man in the middle, such as Eve, would need to solve the ECDLP problem to obtain either **x** or **y** from $\mathbf{k_A}$ or **r**.

2.4.1.1 Homomorphism of Elliptic Curve ElGamal

As mentioned above in section 2.4.1, one of the reasons the Elliptic Curve version of ElGamal was chosen as the encryption system for the InnerCircle protocol for this thesis is because it is ***additively homomorphic***. This means that when you have two values encrypted using Elliptic Curve ElGamal with the same key **k** and you add those encrypted values together, the result of the addition is the same as if the addition was done using plaintext values, only that now the result is also encrypted, i.e. $D_k(E_k(m)+E_k(n)) = D_k(E_k(m+n))$.

Following is the proof of this attribute for Elliptic Curve ElGamal. Assume m_1 and m_2 are already mapped to the subgroup G:

Addition of the two encrypted values becomes:

$$E_k(m_1) + E_k(m_2) = (r_1, m_1 + ky_1) + (r_2, m_2 + ky_2) = (r_1 + r_2, (m_1 + ky_1) + (m_2 + ky_2))$$

The decryption then becomes:

$$\begin{aligned} D_k(E_k(m_1) + E_k(m_2)) &= (m_1 + ky_1) + (m_2 + ky_2) + (-(r_1 + r_2)x) \\ &= (m_1 + gxy_1) + (m_2 + gxy_2) + (-(gy_1 + gy_2)x) \end{aligned}$$

$$= m_1 + gxy_1 + m_2 + gxy_2 - gxy_1 - gxy_2 = m_1 + m_2$$

Compare this to encryption and decryption of two already added values:

$$E_k(m_1 + m_2) = (r, (m_1 + m_2) + ky)$$

$$\begin{aligned} D_k(E_k(m_1 + m_2)) &= ((m_1 + m_2) + ky) + (-(rx)) = ((m_1 + m_2) + gxy) + (-gxy) \\ &= (m_1 + m_2) + gxy - gxy = m_1 + m_2 \end{aligned}$$

Thus one can observe that the criteria for the additively homomorphic attribute holds, i.e. that $D_k(E_k(m_1) + E_k(m_2)) = D_k(E_k(m_1 + m_2))$ is true, for Elliptic Curve Elgamal.

2.5 Identity based encryption(IBE)

The idea of Identity Based Encryption(IBE) was first put forward in 1985 by A. Shamir in the paper "Identity-Based Cryptosystems and Signature Schemes" as an alternative to the key based public-key systems [8]. Instead of relying on a key pair consisting of a public- and private-key for encryption and decryption the users would be issued a secret decryption key based on their identity by some trusted key generator [8]. The idea is then that the input for the encryption would be the message to be sent and the targeted recipients identity and that the resulting ciphertext only can be decrypted using the secret key issued to the recipient.

One of the advantages of an Identity Based Cryptosystem, over a "traditional" public-key system, is that it does away with the need for key-registers, since a user does not need to keep track of public encryption keys to send encrypted messages [8]. All someone needs to encrypt a message to a specific person is their identity. Assuming of course that the recipient is part of the same cryptographic system, because it does not matter if the sender wants to use IBE to encrypt the messages if the recipient only uses RSA for example.

Another advantage of IBE is that the system is independent of the key generator [8]. After the key generator has created the secret key for a user it never needs to be involved again. This is different from the key-based systems where, as discussed previously in the thesis, there are a number of Certificate Authorities(CA's) that need to keep track of if a certain public-key truly belongs to a certain user and issue the official certificates for each public-key <-> user combo.

These CA's remain as passive participants for the entire public-keys lifetime as each time it's corresponding certificate is fetched by someone wanting to encrypt a message using the key, they need to be able to authenticate the certificate. However, a great weakness with having to use a trusted key-generator is just that, it has to be ***trusted***. This trust is paramount for the system since if the key generator is malicious it can be in possession of all the keys it has generated [10]. The key generator can thus decrypt every single message directed to a user whose secret key was created by it [9].

However, in his paper A. Shamir could only provide a concrete idea for how to create signatures using an identity based system and not an encryption/decryption system. His ideas were also based around the uses of smart cards that would provide the functionality and be issued by the key-generator [8]. It took until 2001 before an Identity Based Cryptosystem was suggested that was practically usable.

2.5.1 Boneh and Franklin IBE

As stated above in 2.5, Boneh and Franklin where the first to suggest a practically usable Identity Based Encryption Scheme which is a type of ECC encryption system utilizing the *Weil Pairing* [9]. They suggest a system built from a *bilinear map e* such that $e: G_1 \times G_1 \rightarrow G_2$, between two groups G_1 and G_2 with their security based on a variant of the *Computational Diffie-Hellman assumption* they call the *Bilinear Diffie-Hellman assumption* [9].

Boneh and Franklin define the following properties that must be fulfilled by the bilinear map $e: G_1 \times G_1 \to G_2$ for it to be used in their system [9]:

Bilinear: $e: G_1 \times G_1 \to G_2$ is bilinear if, for all $P, Q \in G_1$and all integers *a, b*, $e(aP, bQ) = e(P, Q)^{ab}$ holds true [9] [19].

Non-degenerate: Not all pairs $(P, Q) \in G_1 \times G_1$are sent to G_2 through the map [9] [19].

Computable: $e(P, Q)$ can be efficiently computed by an algorithm for all $P, Q \in G_1$[9].

Boneh and Franklin call a map fulfilling the above properties an *admissible* bilinear map and the show that the Weil Pairing over elliptic curves constitutes such a map [9]. Understanding what the Weil Pairing is and how they use it to build their system is not needed for this thesis and lies outside the scope of this thesis, interested readers are thus directed to their paper [9] for a closer explanation and proof of the concepts that are presented.

As previously stated, the security for the Boneh and Franklin IBE is based on a variant of the computational Diffie-Hellman assumption called the Bilinear Diffie-Hellman(BDH) assumption and is the assumption that the Bilinear Diffie-Hellman problem is hard to solve [9].This problem is defined as: Given (P, aP, bP, cP) compute $e(P, P)^{abc}$, where $e: G_1 \times G_1 \to G_2$ is a bilinear map fulfilling the earlier mentioned criteria, G_1 and G_2 are groups of order q, q being a prime. The numbers a, b and c are real numbers smaller or equal to q and P is a generator for G_1. For reference, the Computational Diffie-Hellman problem is to calculate abP in G_1 given (P, aP, bP) [9] [19]. Boneh and Franklin describe a BDH parameter generator that is used to generate the parameters G_1, G_2, q and e. See their paper [9] for a description of such a generator.

The IBE scheme consists of four algorithms: **setup, extract, encrypt** and **decrypt**. The setup algorithm generates the global system parameters, like in any ECC system and creates a **master-key** [9]. Extract is used to create the private keys for users corresponding to their IDs, while the encryption algorithm encrypts using public ID's as keys and decrypts using the corresponding private key [9]. The extract and setup algorithms are performed by a *Private Key Generator(PKG)* and the encrypt and decrypt algorithms can be run by anyone in the system [9].

Boneh and Franklin describe two ways of implementing the above algorithms called **BasicIdent** and **FullIdent** [9]. The difference between them is that BasicIdent is not *chosen ciphertext secure* [9], i.e. it is possible to efficiently compute the private key if the attacker can provide the system with any number of ciphertexts and be provided with the corresponding plaintexts. However, this type of security is not of interest in regards to IBE in this thesis and thus the BasicIdent implementation is what will be described below, instead of the FullIdent implementation.

The algorithms setup, extract, encrypt and decrypt works in the following way in the BasicIdent scheme [9] [10]:

Setup:

1. Run the BDH generator with a security parameter k, this generator then generates the prime q, the two groups G_1 and G_2 of order q and the bilinear map *e*. The size of the prime q is decided by the security parameter k. Choose a random generator P for the group G_1.
2. Pick a random integer *s* that is smaller than q and set the P_{pub}= sP.

3. Choose a hash function that hashes from bits to G$_1$, $H_1: \{0,1\} \rightarrow G_1$ and a hash function that hashes from G$_2$ to *n* number of bits, $H_2: G_2 \rightarrow \{0,1\}^n$. Both of these hash functions have to be cryptographic hash functions.

The global system parameters are thus $params = (q, G_1, G_2, e, n, P, P_{pub}, H_1, H_2)$ and the master-key is *s* [9] [10].

Extract: Given a string ID in bit form the following happens:

1. Compute $Q_{ID} = H_1(ID)$, $Q_{ID} \in G_1$.
2. Set the private key $d_{ID}$to $d_{ID} = sQ_{ID}$.

Encrypt: Encrypting a message *M* using the ID is performed by doing the following:

1. Compute $Q_{ID} = H_1(ID)$, $Q_{ID} \in G_1$.
2. Choose a random integer *r*.
3. The ciphertext is computed as $C = (rP, M \oplus H_2(g_{ID}^r))$ where $g_{ID} = e(Q_{ID}, P_{pub}) \in G_2$and $\oplus$is the bitvise xor function.

Decrypt: To decrypt Cusing d_{ID}compute the following:

$$M = (M \oplus H_2(g_{ID}^r)) \oplus H_2(e(d_{ID}, rP)).$$

For this cryptography scheme to work, i.e. that $D(E(M)) = M$, then $g_{ID}^r = e(d_{ID}, rP)$ must be true. This is because the xor function is used twice and $(A \oplus B) \oplus B = A$ for the xor function and with decryption being $M = (M \oplus H_2(g_{ID}^r)) \oplus H_2(e(d_{ID}, rP))$, $H_2(g_{ID}^r))$ must equal $H_2(e(d_{ID}, rP))$ for the computation to be true. The equivalence of $g_{ID}^r = e(d_{ID}, rP)$ will be shown below:

$e(d_{ID}, rP) = e(sQ_{ID}, rP) = e(Q_{ID}, P)^{sr} = e(Q_{ID}, P_{pub})^r = g_{ID}^r$[9]. Note that $e(sQ_{ID}, rP) = e(Q_{ID}, P)^{sr}$ is the bilinear property of e[9].

Providing the proof of security of this scheme is outside the scope of this thesis. For those interested in said proof, is the paper from Boneh and Franklin [9].

Boneh and Franklin also discuss a scheme utilizing asymmetric pairings instead to be able to use slightly more general bilinear maps [9]. This scheme mildly changes the above described scheme in the following ways:

1. Instead of $e: G_1 \times G_1 \rightarrow G_2$ the map is $e: G_0 \times G_1 \rightarrow G_2$ with all three groups being of prime order q.
2. P and P$_{pub}$ are elements of G$_0$ instead of G$_1$.
3. $H_1: \{0,1\} \rightarrow G_1$ instead $H_1: \{0,1\} \rightarrow G_0$, note that in the earlier scheme G$_1$ represented the first group, which makes it equal to G$_0$ in this scheme.

They also slightly need to alter the BDH assumption to the co-BDH assumption [9]:

With random $P, aP, bP \in G_0$and $Q, aQ, cQ \in G_1$no polynomial time algorithm can compute $e(P, Q)^{abc}$with measurable probability. $e: G_0 \times G_1 \rightarrow G_2$ is asymmetric since the elements in the pair are members of different groups [9].

2.5.2 Distributed Identity based encryption

As mentioned earlier in section 2.5, in an Identity Based Cryptosystem (IBC) the Private Key Generator (PKG) can decrypt all the ciphertexts in the system [10]. This is thanks to the inherent key escrow that arises from the fact that all secret keys are created using the master-key belonging to the PKG [10]. However, Boneh and Franklin propose that distributing the key generation in their IBE scheme would solve this key escrow issue [9] [10].

They suggest dividing up the master-key among n nodes and make it so that any set of those n nodes of size t or smaller cannot compute the master-key [9] [10]. However, a user can extract their private key by obtaining t+1 or more shares of a private key [9] [10]. This type of distributed PKG is called a (n, t)-distributed PKG.

While Boneh and Franklin only hint at how to create a complete distributed approach for the PKG, the authors Kate and Goldberg proposed an implementation of distributed Boneh and Franklin IBE(BF-IBE) in their paper “Distributed Private-Key Generators for Identity-Based Cryptography” [10] and provide a proof of security for their solution. Their proposed solution requires a bootstrapping procedure and assumes an asymmetric pairing, following is the said bootstrapping procedure for a BF-IBE cryptosystem [10]:

1. Decide the size n of the node group and the security threshold t, determine these such that $n \geq 3t + 1$ for asynchronous case and $n \leq 2t + 1$ for the synchronous case.
2. Run the BDH generator mentioned in the BF-IDE section to generate e, G_0, G_1 and G_2.
3. Decide on two generators $P_0 \in G_0$ and $P_1 \in G_1$. These are used for generating the public parameters.

In an IBE system the PKG role ends after the key generation is done [8]. This leads to only the previously explained **setup** and **extraction** algorithms being changed in BF-IBE to accommodate a distributed PKG [10]:

Setup: Each node N_i in the set of n nodes participates in a Distributed Key Generation (DKG) over the field Z_q, i. e a field of all integers smaller than q, to generate its share s_i of the master-key s. The system public key is then a tuple of the form $(sP, s_1P, \dots\dots, s_nP)$[10]. Compare this to the public-key in the non-distributed case, $P_{pub} = sP$, and it is plain to see that the distributed case is simply a tuple of the non-distributed public-key and what can be viewed as each node's “personal share” of the public-key. Kate and Goldberg write that this tuple is obtained using a protocol based on a DKG protocol created by Pedersen [23].

Extraction: The user wanting to extract their key contact $2t + 1$ generators for shares. Only $t + 1$ of the received shares need to be correct for the user to be able to construct the key [10]. After the user with the identity ID has contacted all of the $2t + 1$ nodes, all honest contacted nodes authenticate the ID and returns the key share as $d_{IDi} = Q_{ID}s_i = H_1(ID)s_i$. The user can then construct their key as $d_{ID} = \prod_{i=0}^{2t+1}(Q_{ID}s)\lambda_i$, where λ_i is the Lagrange coefficient [10].

From the above it can be observed that once the extraction algorithm is complete, everything needed to perform encryption and decryption as in BF-IBE is in place and there is no longer any difference between the distributed and non-distributed versions. Just as for the previously discussed BF-IBE, for proof of concept and proof of security for this distributed version, see the paper by Kent and Goldberg [10].

Note that the difference between the formulas above and those in the paper [10] is purely cosmetic, all descriptions put forward of IBE in this thesis is done using elliptic curves and, as previously mentioned, Ps for elliptic curves is analogous to P^s in Z.

3. Problems and Approach

This chapter is where the research question for the thesis is laid out and what the challenges said question entails are discussed. The approach taken to tackle the research question is also presented as to which type of research method was followed and how it was utilized in a manner that was most conducive to this thesis.

3.1 Challenges

There are two main challenges in this thesis, solving the problem of confidentiality without introducing scalability issues.

3.1.1 Confidentiality Challenge

There are three main attributes in Information Security, these are confidentiality, integrity and availability. Location proximity protocols focus on confidentiality, they provide users with the ability to use location based services without “leaking” their actual location [1]. This means that the aim of these protocols is to keep the exact location of their users hidden, it is confidential information.

However, in a decentralized location proximity protocol, such as InnerCircle, another problem with confidentiality arises. This problem is that there is no way to ascertain who the sender is when a message is received. This could lead to a user modifying their own messages, breaking integrity, to be able to perform the location proximity with another user whom they are otherwise not authorized to do so with, breaking confidentiality, by pretending to be someone else. For example, we have three people, Alice, Bob and Eve. They are all users of some location based service used to find out if friends are close to them, using InnerCircle. Alice and Bob are friends while Eve is Bob's stalker. Bob would thus have allowed the user “Alice” to find out if Bob is close to her, but not the user “Eve”. However, Eve could modify her instigating message in such a way that she pretends to be the user “Alice” and thus be allowed to find out if Bob is close to her, as illustrated in figure 11 below.

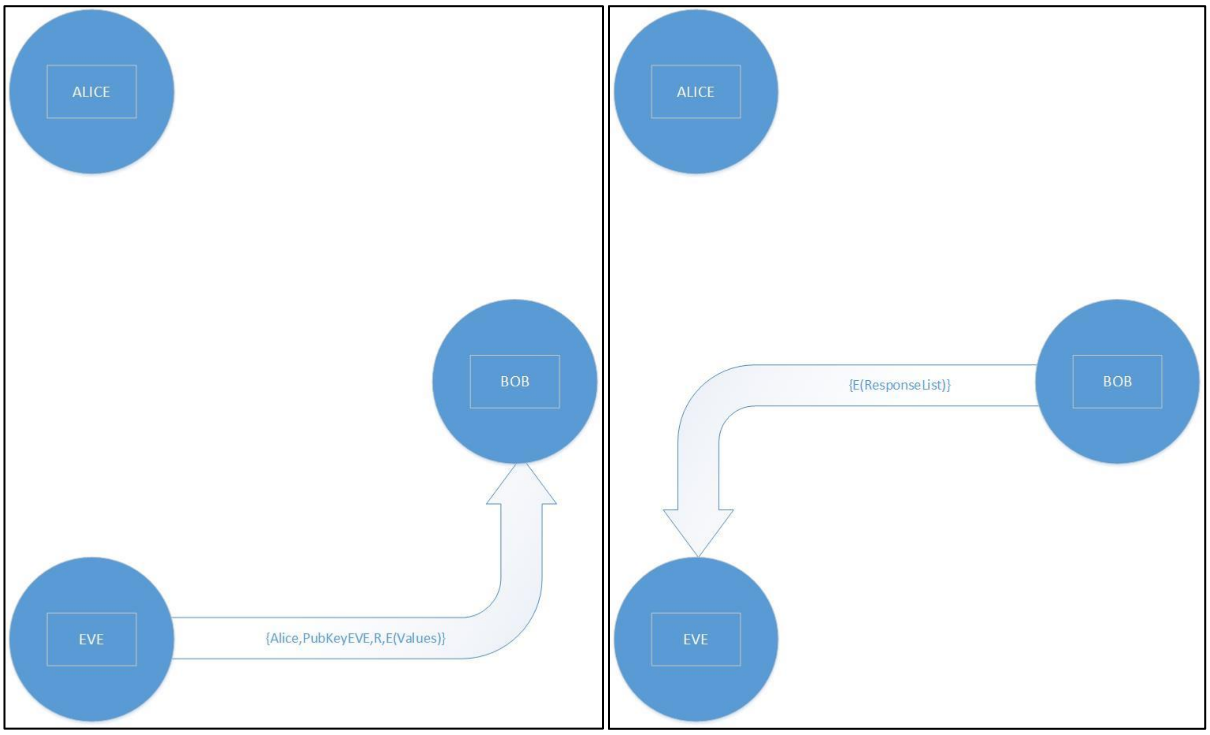


Figure 11. (a) The client with id Eve is blocked from performing location proximity with the client with id Bob. Eve pretends to the client with id Alice to try and circumvent this block. (b) Since there is now for Bob to confirm the authentication of the id, key pairing of (Alice, PubKeyEVE) he has no choice but to perform the location proximity and respond to the sender, which in this case is actually Eve and not Alice.

The problem is that InnerCircle is, as said earlier in section 2.2.4, a decentralized protocol to be used in a decentralized setting, meaning that there is no public space for users to put and extract keys. This leads to that each time a user, Alice, wants to check if her friend, Bob, is within 100 meters of her, she needs to send her public-key to Bob along with her (encrypted)location information. This is because she is the instigator the calculations must be done under her public-key. However, there is also no clear way for Bob to confirm that the instigator of the exchange is actually Alice. It could be Eve, his stalker as mentioned earlier, having modified her instigating message to look like it comes from Alice, but with Eve's information and public-key instead.

In a centralized setting this would not be an issue, not only is the need to submit the public-key with every message removed, in a centralized setting there also exists Certificate Authorities(CAs) [25]. Why we cannot simply store all keys from all users in the decentralized case will be explained later in 3.1.2. In the usual centralized public-key cryptosystems every public-key is provided in a certificate, this certificate contains not only the public key, but also the identity of the owner and is signed by a trusted third party, a CA, that confirms that the key truly belongs to said person [25]. In this type of setting Bob could simply check the certificate containing the key to establish that if it truly is Alice who instigated the protocol with him and not his stalker Eve, maintaining confidentiality.

However, in a decentralized setting, the CAs do not exists, this is because CAs are centralized authorities and are not allowed to exist in the decentralized setting. Thus Bob cannot use a certificate to ascertain the identity of the owner of the key he has received, he simply has to trust that the sender is who they say they are. This does not ensure confidentiality. In the InnerCircle paper [1] the authors suggest that two users, Alice and Bob, could use some other channel to exchange certificates. However, this does not solve the problem in the decentralized setting as exchanging the certificates beforehand is a centralized solution, since CA's will be used to verify them. It also requires a third-party solution to solve the problem. The aim of this thesis is to solve this problem fully decentralized without requiring the use of such a third-party solution to provide a standalone solution.

Now, this problem might be solved if as soon as a new user is added to a location based service implementing InnerCircle, or protocols like it, their public key is sent to all other users whom then save that key together with the new user, binding key and user together at all other users. That solution would introduce a (new)scaling problem.

This problem can also be seen as a type of authentication problem.

3.1.2 Scaling Challenge

Before we get into the scalability issues we would like to avoid, we will mention that there is a scalability issue already inherent in decentralized privacy-preserving location proximity protocols, one concerning network resources. If users need to keep resending their keys, then if 1000 users where to perform the location based service with all other users in the system at the same time, the network resources required would be ~2 Gb/s, 1000*1000*2048. This is because the recommended key size for most public key cryptosystems is 2 Kb. We do not attempt to solve this problem, instead we attempt to solve the confidentiality problem without introducing other scalability problems.

Simply storing all keys would introduce another scalability issue. This scalability problem would be one simply of storage. For a small system the amount of storage to keep all keys is not very high, for a 1000 users it would be ~2 Mb(1000 * 2048). But as the number of users increases so does the storage demands. If instead there were 1 million users, each device would need to devote 2 Gb of storage instead, just for storing keys.

The final scaling challenge is that the solution to solving the confidentiality problem in the system should have as low of an impact as possible on the performance on the underlying decentralized privacy-preserving location proximity protocol, in this case InnerCircle. This is because if a solution would introduce a large executional scaling problem, i.e. as the system grows, the execution time grows rapidly.

3.1.3 The Research Question

The basis for the Confidentiality Challenge is the lack of authentication in a decentralized system and since a decentralized solution is desired for the InnerCircle protocol the main research question becomes: Is it possible to introduce authentication into the decentralized setting of InnerCircle without adversely affecting the performance or worsen the scaling? The above question is the one we will try to answer through this thesis.

3.2 Approach

The intended approach is to take the decentralized location proximity protocol InnerCircle and change the first message and initial behavior of the protocol to provide it with the ability to spread public keys between the users and the validation functionality provided by certificates in centralized systems.

Two different solutions where explored in the attempt to modify the original protocol, one based mainly on cryptographic techniques and one based solely on techniques found in distributed system.

A small testing application has then been developed which is used to compare the two different solutions with each other, in terms of how much each of them impact the original protocol in an executional time sense.

The data collected will be analyzed to provide an idea how much the two different solutions impact the execution time of the original protocol and how the distributed systems solution’s execution time scales with the size of the system. The data will also be compared to the performance of centralized location proximity protocol, using published performance data of such protocols if available. This is to judge the drop, if any, in performance between a centralized and decentralized solution. If there is a large drop in performance, then it might be so that having to trust a third party, in the case of the centralized solutions, is more acceptable than the performance loss experienced using a decentralized solution.

While we will give suggestion for how to setup each of the solutions and argue why that is correct, the setup will not be part of the testing as we are interested solely in how the execution time is impacted after the users are already in place and utilizing the location based services, thus the time it takes to create and initialize the different environments is not of interest within the scope of this thesis.

3.2.1 Distributed Systems

For the distributed systems solution, called the Gossiping solution, we are going to use the fact that every node in the system has to be connected to at least one other node in the system, since it is a peer-to-peer network [12], to create a neighbor list. Every node will keep the unique identifier and public key of all their neighbors in the neighbor list. Which will contain all, or some, of the other nodes this node is connected to. The decision of how many of the connected nodes is in the neighbor list depends simply on how many nodes each node is connected to.

The idea of using a type of “neighbor” list comes from the fact that it is built into peer-to-peer structured networks to keep track of a set of other nodes close to you to handle nodes entering and leaving the network and managing data sharing in the network. Examples of this can be seen in both Chord and Content-Addressable network [12].

For this approach the changes to the current protocol will be limited to the start as the public key of the claimed instigating party has to be known for the location proximity calculations can be carried out. It is also at the start when the public key is received that the protocol needs to validate the identity of the owner of public key.

3.2.2 Cryptography

For the cryptographic solution we suggest encrypting the answer to the instigator twice, first as usual for the InnerCircle protocol and then a second time using a distributed identity based encryption scheme using a unique identifier for each user in the system to generate the public key needed for the Identity based system.

This solution will also suggest using techniques from distributed systems for the setup for the distributed identity based cryptographic system.

In this approach the changes to the original protocol happens at the end, when the recipient is creating the message to respond to the instigator. Instead of just encrypting the response once using the public-key encryption scheme, they encrypt the result of that encryption again, using the identity based encryption scheme.

Both of these solutions will be described in further detail later in this thesis.

3.2.3 Research method

The research method used for this thesis can be viewe as a variant of the scientific method, which is taking a solution, testing it and trying to improve upon it. However, instead of going through a testing -> improvement cycle for a single solution I have started with a base solution, the Gossiping solution, and developed it further into a solution based more in Cryptography. Since these solutions differ in which field they are primarily based, they are tested against each other to see how each performed. This is because the Cryptographic solution does not so much improve on the Gossiping solution as changing the core method of providing authentication, or confidentiality, whilst still maintaining parts of the Gossiping solution.

4. Solutions

This Chapter is divided into two parts, dedicated to one solution each. For each solution the underlying idea, how the solution would work, a possible implementation, which challenges it solves and how it affects the scaling, the security consideration for the solution in question and which changes would have to be made to the InnerCircle protocol for the solution to work is covered. This chapter also utilizes illustrated examples to show of the solutions would work.

4.1 Gossiping

4.1.1 The Idea

The idea for the gossiping solution is to construct the system as an unstructured distributed network and that each time a location proximity request is received the receiving node floods the network with queries if a received (ID, KEY) pair is not recognized.

4.1.2 How it works

In an unstructured network each node keeps a neighbors list of randomly chosen nodes in the network [12], as previously explained in section 2.1. This solution utilizes this fact by each node storing a (ID, KEY) pair for each of its neighbors. Whenever a location proximity request, which contains the sender's ID, their public key and their encrypted location values, is received the receiver checks if any of the (ID, KEY) pairs it has stored contains the ID received in the request. If it finds the ID in its own neighbor list it can confirm if the KEY is correct and either carry on with the procedure or reject the request. If it does not find ID in its own list, the node sends a (ID, KEY) question request to all its neighbors, these neighbors then check their lists for the ID and if found confirms if the KEY is correct and responds with the result. If a node does not find the ID in the (ID, KEY) question in their list, they forward the question to all its neighbors except for the neighbor it received the question from, this is to avoid unnecessary network traffic. Eventually all neighbors of the node with id ID will have been contacted and confirmed if the KEY is correct or not and the original receiver will have either carried out the procedure or rejected the request using the responses from IDs neighbors.

To make sure that a single misbehaving node cannot alter the behavior of the receiver, as in providing a false result which will result in the receiver wrongly reject or accept a request, a security threshold *k* is used. This security threshold requires *k* of the *m* neighbors of ID to respond with a positive result for the request to be accepted. If the minimum number of neighbors required, *m*, is higher than the number of nodes currently in the system then *k* of the current number of nodes in the system are needed for a positive response. This solution assumes a majority honest network and that each node *n* has roughly the same amount of *m* neighbors. The solution also assumes that a majority of each nodes neighbors are honest. Otherwise, a situation could arise in which all the neighbors of some node *n* are malicious and providing false responses regarding that particular node, whilst the majority of the nodes in the network are still honest.

For the reason of avoiding more network traffic than necessary, if a node receives the same (ID, KEY) question with the same original sender, i.e. the node that received a location proximity request, within a certain amount of time, the question is dropped instead of checked for and forwarded if needed. This is to stop the possibility of the entire network endlessly circulating the same question and make sure that each node only handles the same question once. The original sender of the (ID, KEY) question should also drop the question if queried with their own question. This is done by the help of each message keeping track of who the original source of the question is. As has been mentioned multiple times all the responses are sent to the original sender, which means that the actual look of the message is closer to {(ID, KEY)?, SOURCE:BOB} for example where (ID, KEY)? is the question and the SOURCE:BOB indicates who is posing the question. This is to make sure that if two or more nodes all ask the same question they all receive correct responses.

Below, in figures 12-15, is an illustrative example of this solution with a threshold *k* set to 60%, i.e. m*0.6 of Alice’s neighbors must respond positively for the request to be accepted.

With m being the total amount of her neighbors:

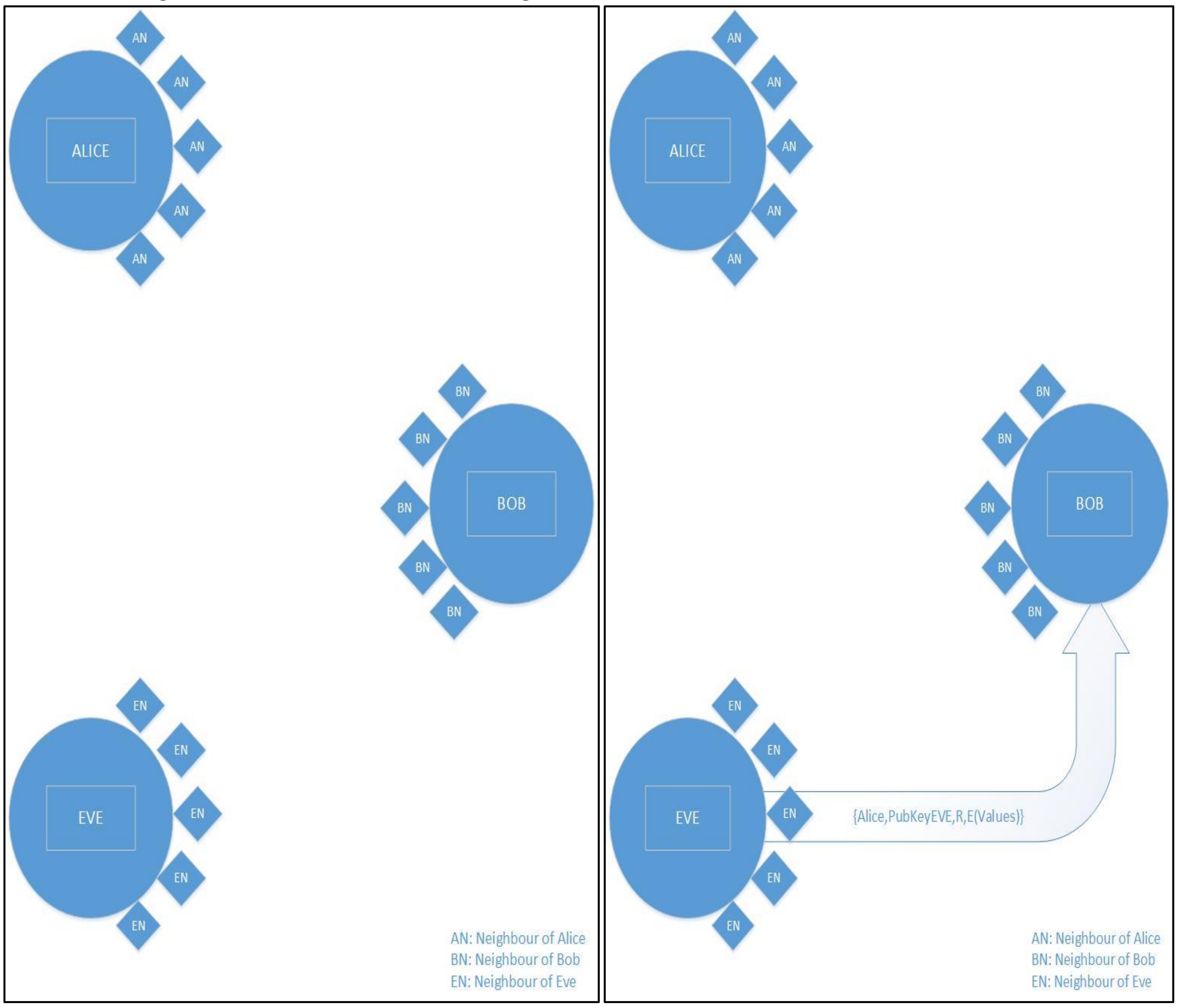


Figure 12. Step 1 (On the left). The setup of the network is shown the left picture with a focus on the clients Alice, Bob and Eve and their immediate neighbors. All other clients are "invisible" for clarity. Step 2 (On the right). In the second picture we see Eve trying to perform Location proximity by pretending to be Alice, by submitting Alice as the Id, with the key and values being those of Eve. R is the distance to check against.

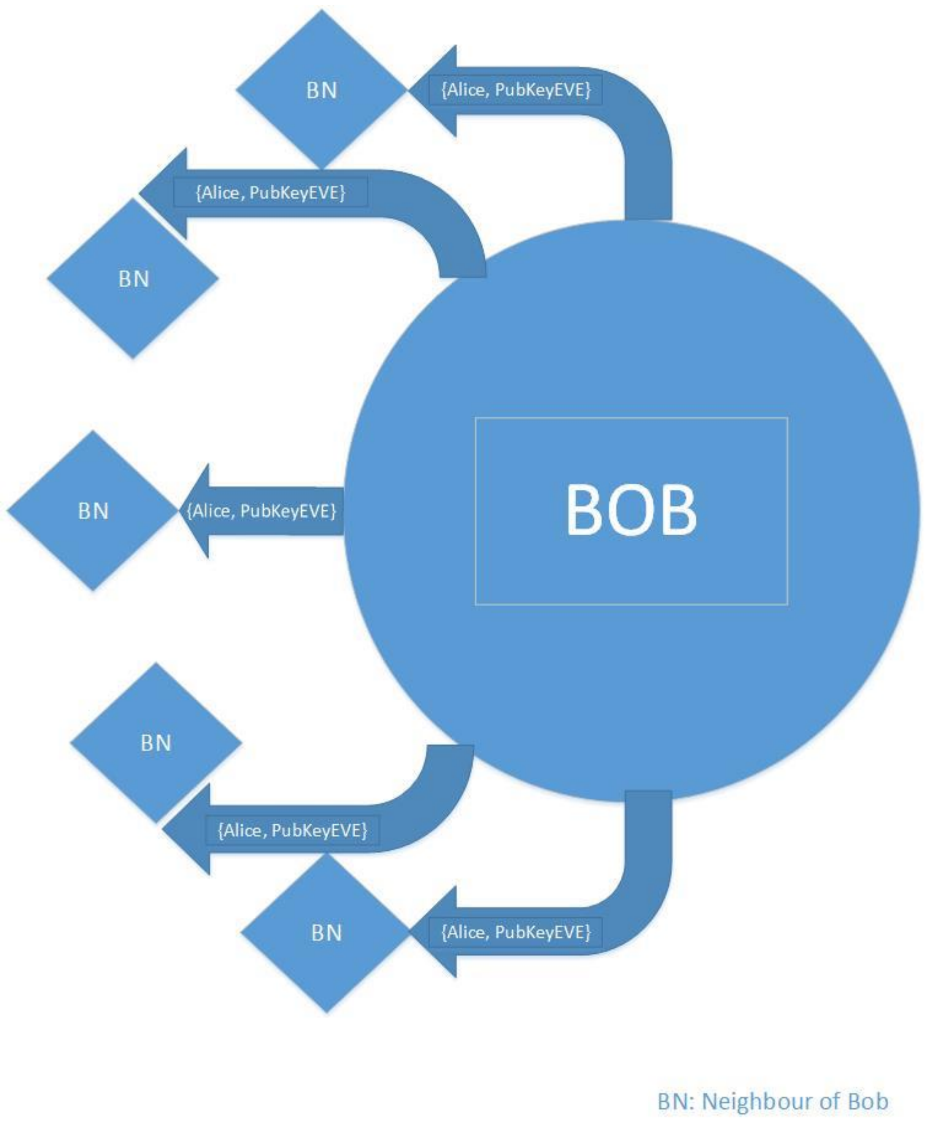
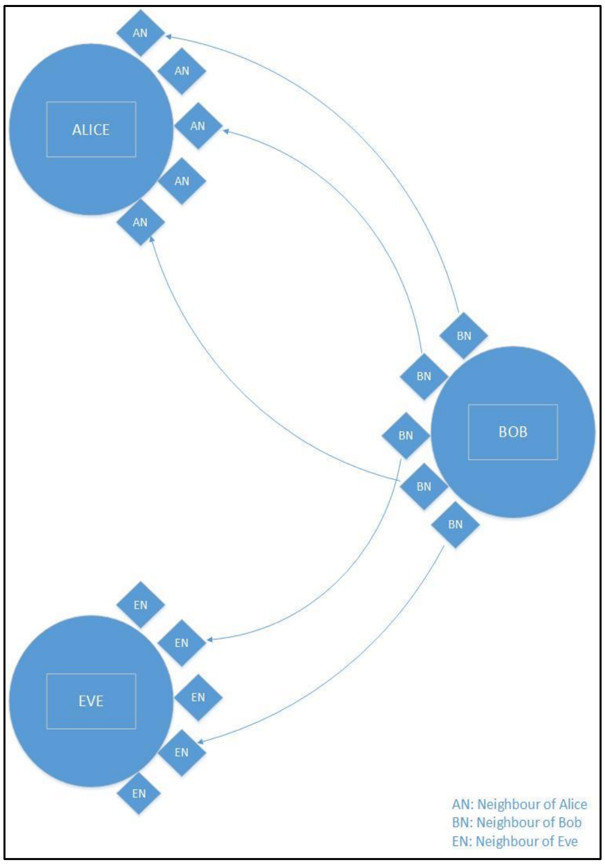
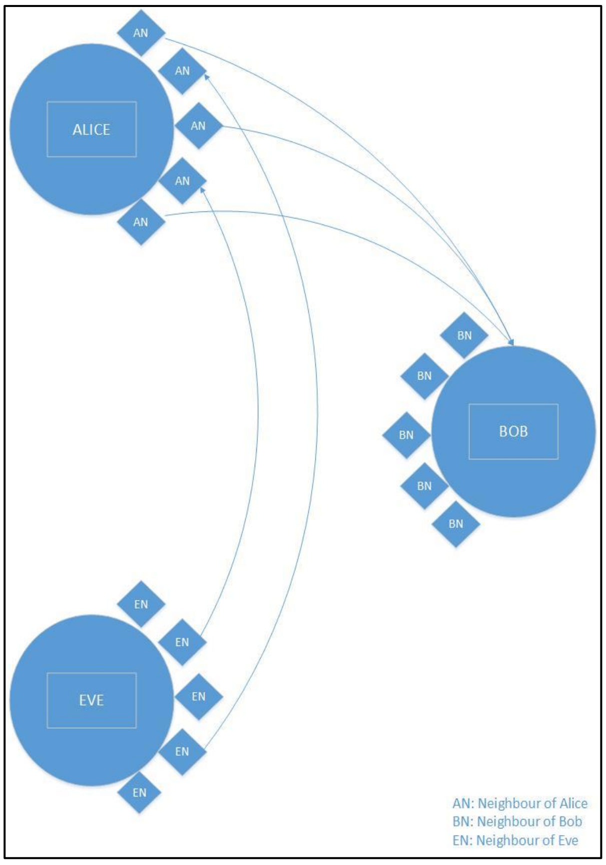
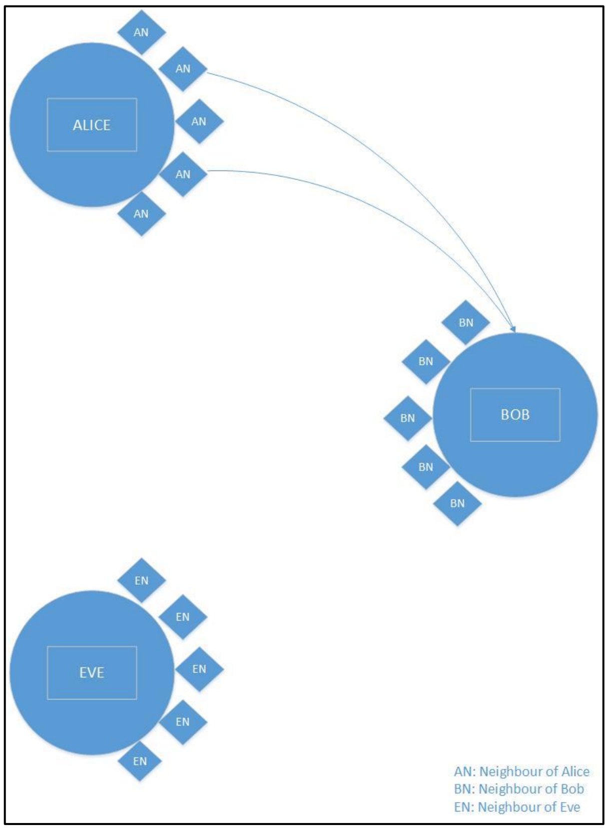


Figure 13. Step 3. Having received the query from Eve, pretending to be Alice, Bob confirms that he is not a neighbor of Alice and thusly sends a confirmation query to all his neighbors to confirm if the key PubKeyEVE belongs to the id Alice.

Figure 14. Step 4-6. In the below three pictures the forwarding from Bob's neighbors is demonstrated. (a) Firstly, in the left picture Bob's neighbors have confirmed they are not neighbors with Alice and thus forwards the query from Bob to all of their neighbors. Only the forwarding to neighbors visible in the picture is shown. (b) Secondly, in the right side picture, those clients that are neighbors with Alice and received the forwarded query responds to Bob by either confirming the (ID, KEY) pair or denying it. The clients that also received the query but are not neighbors with Alice keeps forwarding the query from Bob. (c) Finally, in the last picture all clients that are neighbors with Alice have now received the query from Bob and responded to him by either confirming or denying the (ID, KEY) pair.


AN
AN
ALICE
AN
AN
AN
BN
BN
BOB
BN
BN
BN
EN
EN
EVE
EN
EN
EN
AN: Neighbour of Alice
BN: Neighbour of Bob
EN: Neighbour of Eve


AN
AN
ALICE
AN
AN
AN
BN
BN
BOB
BN
BN
BN
EN
EN
EVE
EN
EN
EN
AN: Neighbour of Alice
BN: Neighbour of Bob
EN: Neighbour of Eve


AN
AN
ALICE
AN
AN
AN
BN
BN
BOB
BN
BN
BN
EN
EN
EVE
EN
EN
EN
AN: Neighbour of Alice
BN: Neighbour of Bob
EN: Neighbour of Eve

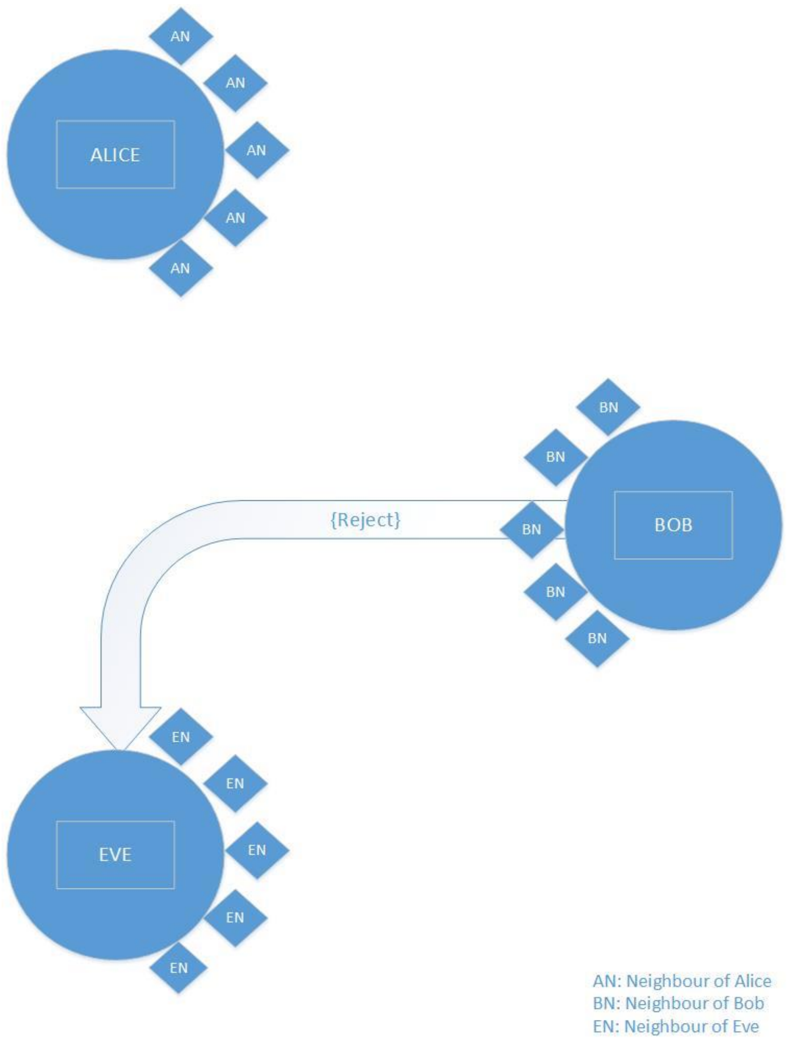


*Figure 15. Step 7. Bob has now received a response from all of Alice's neighbors either denying or confirming the ID, KEY pair and since this pair was Alice, PubKeyEVE and the assumption is majority honesty, the threshold **k** for acceptance will not be reached and Bob responds to Eve rejecting her request.*

4.1.3 What problems does it solve?

Remember that there were three challenges explained to test the solutions against described earlier in section 3.1:

1. Confidentiality (No authentication)
2. Scaling of space required for storing public-keys
3. Impact on the performance on InnerCircle

How the gossiping solution performs in regards to the first two will be mentioned here, with a discussion later that covers *why* it solve those challenges. How it performs in regards to the last challenge will be discussed in the later *Testing and Results* chapter together with performance data in regards to that challenge and compared to InnerCircle [1] without modifications.

4.1.3.1 Confidentiality Challenge

This is solved as long as the majority honesty assumption of the network is fulfilled, any node pretending to be another node will have its location proximity request be rejected after the (ID, KEY) query response falls below the threshold of *k* positive responses.

4.1.3.2 Storage space scaling Challenge
This challenge is solved since each node only need to store the (ID, KEY) pair of its neighbors and not of all nodes in the network. This means that the storage needed will be constant after the initial decision of what the max number of neighbors a node can have in the network is made. The storage space needed for this max number of (ID, KEY) pairs can thus be reserved on the node and will not grow together with the network as would be the case when keeping track of all (ID, KEY) pairs in the system.

However, a note to be made is that the Network scaling issue mentioned in the challenges section, that is not tackled in this thesis, is made worse by this solution, since each single location proximity request floods the network with an exponentially higher amount (ID, KEY) question queries to confirm the validity of the pair.

4.1.4 The changes to InnerCircle for the solution
There would be a few changes to the InnerCircle protocol to modify it for this solution. Firstly, the initial location proximity request message would change from {KEY, ENCRYPTED LOCATION VALUES} to {ID, KEY, ENCRYPTED LOCATION VALUES} for it to be possible to query the neighbors list and, if unsuccessful, query the network to see if KEY belongs to ID. Secondly, the protocol would have to be modified to incorporate the (ID, KEY) querying into the protocol and have it placed before the modified Euclidean distance calculation described earlier. This is to make it possible for a node to reject the location proximity request early in the execution of the protocol.

4.1.5 Suggested Implementation
The focus here will be mainly on how a possible implementation of the unstructured distributed network and the list of neighbors can be done.

As previously covered in section 2.1, an unstructured distributed network topology uses randomization algorithms to randomly select, hopefully live, nodes and place them in a node’s list of neighbors [12]. However, we suggest slightly altering this approach with some of the concepts of the structured distributed system, *Content Addressable Network(CAN)* [13], described in 2.1.1. Each node keeps a tuple representing a d-dimensional Cartesian coordination system and whenever a new node x joins the network it is directed to another node y already in the network and x sends a type of greeting request containing its unique ID and KEY. When node y receives this greeting, it first checks that it has not reached the max number of neighbors and then queries the system checking if ID already exist. If it already exists x is rejected by y and can be either kicked out of the network or requested to pick a new ID before starting over. However, if the ID is not already in the system, y splits one, not already split, dimension in its tuple representation, assigns x to this dimension with x’s (ID, KEY) pair and informs x of this with an accept message containing its (ID, KEY) pair. The new node x then splits one of its dimensions to add y as a neighbor. If *x* does not get a response within some timeframe *t* from *y*, *x* contacts the connection point once more to be redirected to another node. This altered approach is repeated enough times that each new node *n* gets as close as possible to *m* number of neighbors, since *k* of *m* neighbors are needed to respond positively for the location proximity request to be accepted.

Below is a short illustrative construction example in figures 16-18 using three clients A, B and C, with A being the first client.

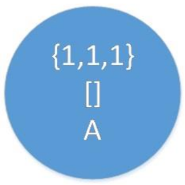


Figure 16. Step 1. Starting point of the system with only a single client A and using a 3-dimensional Cartesian coordinate system (connection point not shown).

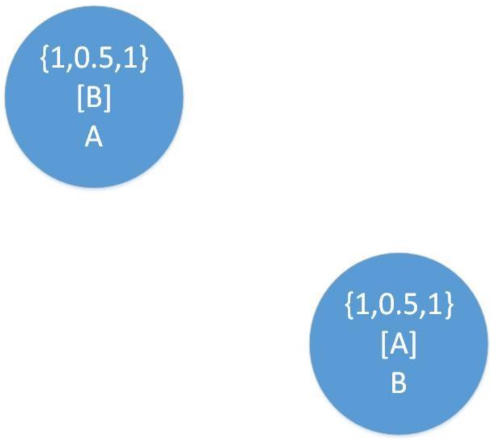


Figure 17. Step 2. Another client B has joined the system and been directed to A. A has thus halved one of its dimension and informed B that they are neighbors and added B to its list. B halves a dimension and adds A to its list of neighbors. It does not matter which dimension B halves but for clarity in the example we halve the same dimension for both.

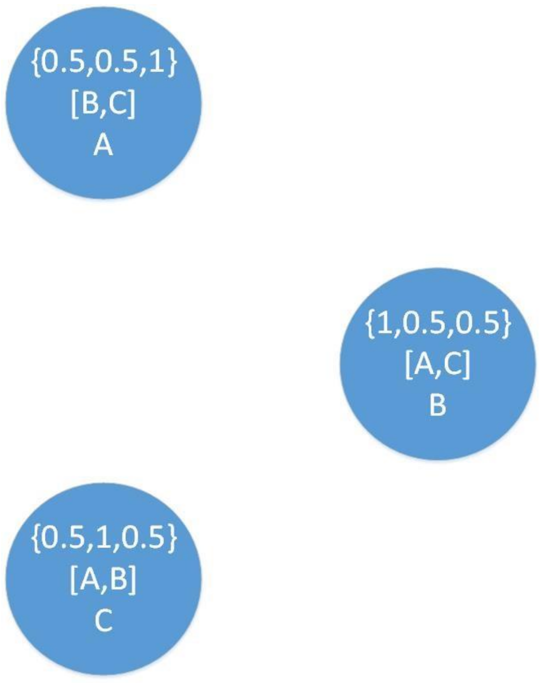


*Figure 18. Step 3. The third client called C joins the network. Firstly, this new client has been directed to either A or B, just as B was directed to A in (Step 2) and the same actions as in (Step 2) are carried out. Secondly, unlike for B in (Step 2), when C goes back to the connection point to get into contact with another neighbor, assuming **m** is not set to 1, C is eventually directed to the other one of A and B it did not become neighbors with the first time around.*

A failsafe can be added to the construction in the case of a client joining the system for the first time and being rejected by all clients already in the system due to the list of neighbors being full for all of them. This failsafe can be in the form of an override flag in the initial query being activated if the new client has gone back to the connection point sufficiently many times without adding a neighbor. The next client to receive the initial neighbor query will, if their list is already full, randomly replace a neighbor with the new client when the override flag is set, informing the replaced neighbor of this.

This implementation also makes sure that the first neighbor *m* of a new node *n* always is a live node, since if *m* is not live then *n* will not get a response from *m* and will thus contact the connection point and try with some other random node.

Another implementation suggestion is to utilize a heartbeat protocol to send messages between neighbor nodes to keep track so that at least one of the neighboring nodes is live and drop any offline from the list. If none of the nodes in the list of neighbors for some node *n* is live, *n* can drop the final, now offline, node in the neighbor list and connect to the connection point once more to redo the construction protocol explained above. For there to not arise confusion with differing neighbor list, if a node *n* goes offline and then becomes live again it redoes the same above construction protocol. See figure 19 for an illustration of this heartbeat.

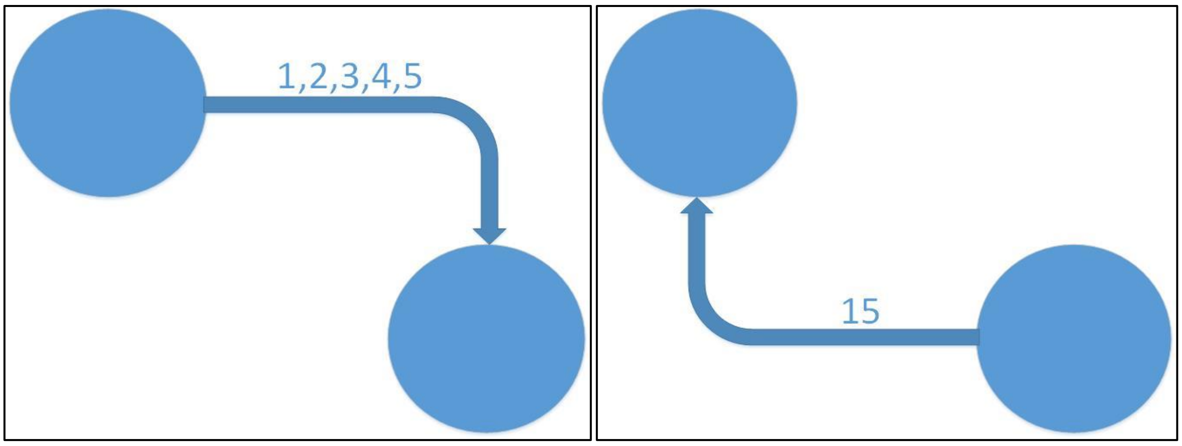


Figure 19. Example of simple heartbeat interaction. The first client wants to check if the second client is alive and thus sends a message containing a few select numbers to the second client. The second client then responds with the sum of the received numbers to show that it is alive.

4.1.6 Security Considerations

We will start by proving that a node cannot exist in the system without being neighbors with at least one other node.

Lemma 1.1: No node can exist in the system without being neighbors with at least one other node.

Proof: Assume that there exists a node *n* that has joined the unstructured network for the location proximity system without existing in any other node *m*'s list of neighbors and thus also having no nodes in its own list of neighbors. This means that the node *n* joined the system without executing the randomization algorithm used to construct the neighbor list for each node that joins an unstructured distributed network, which is the topology used by this location proximity system. This is not possible since the protocol used when joining an unstructured distributed system incorporates the randomization algorithm to construct the list of neighbors. Thus if a node *n* is in the system without being neighbors with any other node *m* in the system it has not executed the joining protocol and cannot possibly exist in the system.

Lemma 1.2: No node can exist in the system without at least being neighbors with one other live node using the suggested implementation.

Proof: Assume that there exists a node *n* that is part of a system created using the suggested construction implementation from section 4.1.5 without being neighbors with any live node *m* in the system. Firstly, for the node *n* to have ever existed in the system at any point in time it must have been, at some point, neighbors with some other node *m*, as previously established. Let us thus assume that for the initial assumption to be true, this node *m* must have either not been live from the very start or gone offline sometime after the initial construction protocol inspired by *CAN* finished. However, as described in the implementation suggestion in 4.1.5, if the node *m* never was live from the very start then *m* would not have responded to *n*'s initial request and thus *n* would have contacted the connection point again and been redirected to some other node *k* to start the neighbor list. This means that for *n* and *m* ever to have been neighbors, *m* would have to have been live at least at the beginning when *n* joined the network and then gone offline sometime after the initial contact. However, as presented in 4.1.5, the heartbeat function for *n* will keep track of if *m* is live or not. If *m* is the only neighbor of *n* and no longer live, then *n* will drop *m* from its list, notice the list is now empty and contact the connection point once more to do the construction protocol again to receive a new neighbor. This leads to that if *n* exists in the system without at least one live neighboring node *m*, then *n* has either not performed the starting construction protocol inspired by *CAN* or it has not been performing the heartbeat protocol and since both of these protocols are part of the suggested implementation and since the system *n* is a part of is created using the suggested implementation from section 4.1.5, *n* must be using these protocols to be part of the system. Thus the original assumption is a contradiction.

An important attribute mentioned previously, in section 4.1.2, is the ability for the nodes in the network to drop (ID, KEY) questions from the system to avoid endlessly circulating a (ID, KEY) question in the network. Failure to do so would quickly surpass the capacity of the network due to the flooding behavior of the (ID, KEY) question query. That this prevention mechanic truly is in place in this solution will be proved by the following proof by contradiction.

Lemma 1.3: No (ID, KEY) question query will be circulated in the network forever.

Proof: Assume that a node *n* forwards the same query *Q* twice. From the view of the node *q*, the node starting the query, all unstructured distributed networks will resemble figure 20.

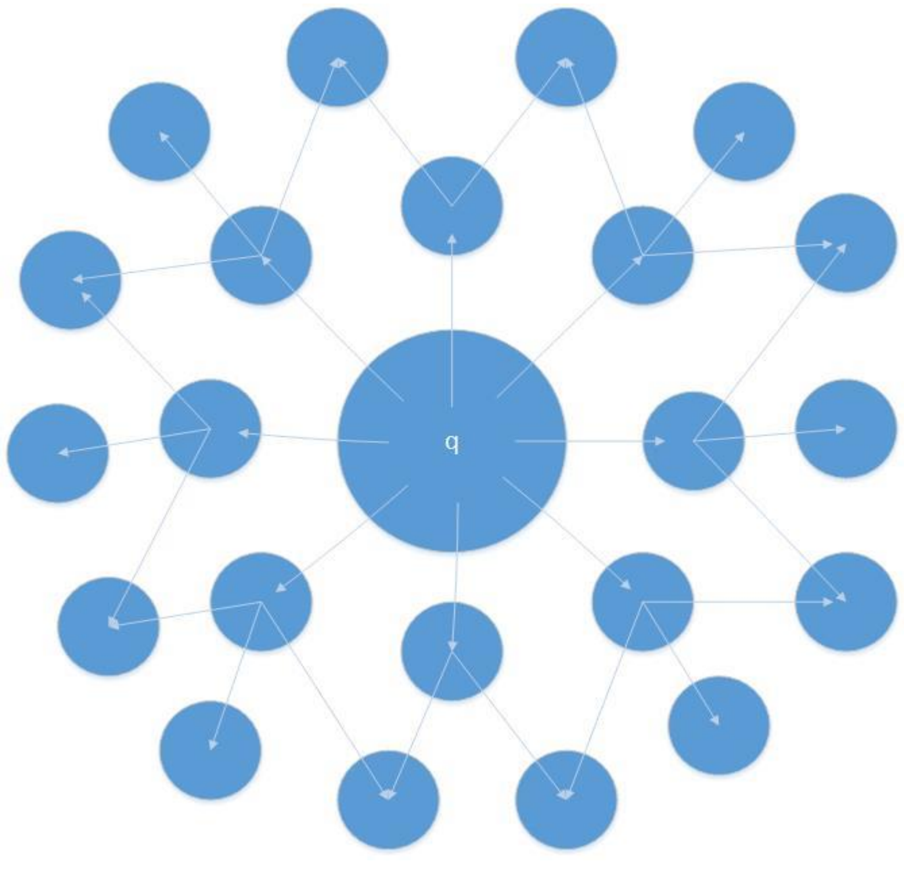


Figure 20. How the networks looks for the node q.

Where the querying node *q* is the node in the center and all other nodes are some distance, or jumps, *d* away. All nodes that have distance $d = 1$ are the neighbors of *q*, all nodes with distance $d = 2$ are neighbors of neighbors etc. However, there will be some overlap where nodes with the distance k are neighbors with other nodes with the distance k, all these nodes are still viewed as nodes with the distance k in the network, since that is their shortest path to *q*. With this view we can safely say that, disregarding latency, all nodes distance $d = k$ will receive the query *Q* before all nodes with the distance $d = k + 1$. With this in mind we can establish that for the opening assumption to be true, the node *n* with distance *k* will have to have received the same query twice. The first query *Q* will have arrived from a neighbor with the distance *k-1*, this is then processed and forwarded if needed by *n*. The second identical query *Q* can have three possible origins:

1. Another neighbor with the distance *k-1*.
2. A neighbor with the distance *k+1* that has another neighbor distance *k*.
3. A neighbor with the distance *k*.

The first two possibilities occur because each node will only completely process one query at a time, in the first possibility this query arrived at the same time as the first one but is processed after and for the second possibility the neighbor distance *k+1* processed the query from its other neighbor with the distance *k* first and thus does not exclude *n* from its forwarding list. For the third possibility the duplicate query *Q* is received since from the viewpoint of this neighbor also distance *k*, *n* has distance *k+1* since n is its neighbor and it will forward the query to all its neighbors except for the one it received the query *Q* from. However, following the protocol, *n* will drop all these queries instead of processing them, since these possible duplicate queries will all arrive shortly after the first one and in this gossiping solution, all duplicate queries that arrive within a short timeframe are dropped. Therefore, the node *n* cannot forward the same query twice and thus the starting assumption is contradicted.

Having covered that any node *n* must be connected to at least one other node *m* if the suggested implementation from 4.1.5 is used and that queries are not endlessly circulated in the system, we will now move forwards with a discussion regarding that the solution provides authentication in a decentralized public-key cryptosystem setting, which is the main objective of this thesis. Starting with proving that there can be no duplicates of identities in the system, since for the solution to accomplish its goals, all id’s need to be unique.

Lemma 1.4: No two clients will have the same unique id.

Proof: Assume that there are two clients in the system with the same id.

For the above assumption to be true, the following must have happened:

1. A client with id *ID* joins the system.
2. The connection point redirects the client to some other node *n* in the system.
3. Node *n* performs the rest of the joining protocol described in the implementation suggested in section 4.1.5 and adds the node of the client with id *ID* as a neighbor.

4. Steps 2 and 3 are repeated until the node of the client with id *ID* has a sufficient amount of neighbors.
5. Another client with the same id *ID* joins the system.
6. Steps 2 through 4 are repeated for this new client.

However, the process will be stopped in step 3 for this new client with the same id *ID*, since, as covered in 4.1.5, whenever a node *n* receives a greetings message from a node of a new client, it queries the system to make sure the id *ID* does not already exist in this system. Thus setup 6 above will not be completed and there will be no two clients with the same id *ID*, the starting assumption is a contradiction.

Theorem 1: No malicious client(Eve) can pose as some other client(Alice) to conduct location proximity with a third client(Bob) who has blocked Eve from conducting location proximity with him.

Proof: Assume that Eve can purposely alter her request message to Bob from {Eve, PubKeyE, $E_{PubKeyE}$ (x, x^2, y, y^2), r} to {Alice, PubKeyE, $E_{PubKeyE}$ (x, x^2, y, y^2), r} to appear to be the client Alice to Bob and that Bob accepts this modified message and performs location proximity. Let's also assume that Bob and Alice are not neighbors. For this to be true the id, key pair for (Alice, PubKeyE) must have been confirmed through the use of a flooded query question as described earlier in section 2.1, since Bob is not neighbors with Alice he will have queried his neighbors for confirmation. However, as the query floods through the system it will eventually reach the neighbors of Alice, since, as previously established, she is connected to at least one other live node in the system, and these neighbors will send a message *rejecting* the id, key of {Alice, PubKeyE} as the key they have connected with the id Alice is PubKeyA, not PubKeyE. This will lead to Bob rejecting, and not accepting, Eve's request. Therefore, the assumption is a contradiction and thus Eve cannot pose as Alice to conduct location proximity with Bob.

The above theorem and accompanying lemmas shows that the gossiping solution achieves the main goal of providing public-key authentication in relation to owners in a decentralized setting where certificates are of no use due to the lack of certificate authorities. However, this authentication is the only function of certificates and certificate authorities that this gossiping solution can substitute for in a decentralized setting. There is no way for this solution to handle certificate revocation nor can it handle key validation, i.e. make sure that the key is still in use, that the client has not switched key, among other functions.

4.2 Cryptography

4.2.1 The Idea

The idea behind the cryptographical solution is quite simple, encrypt the response once more. As previously established, InnerCircle does all the calculations on encrypted values using the homomorphic attributes of some encryption scheme like Elliptic Curve Elgamal [1], this leads to the response already being encrypted once using the homomorphic scheme. However, to stop a malicious node from pretending to be someone else, the response could be encrypted again using Identity Based Encryption scheme which uses the ID incorporated in the request message as key. Thus is if a node pretends to be someone they are not, they will not be able to decrypt the response as they are not in possession of the decryption key for ciphertexts encrypted under someone else's ID. While this would not stop a node from trying to act as another user, they will not be able to decrypt the response and thus not know if the receiver of their location proximity request is within the distance *r*.

4.2.2 How it works

This idea works similar to the gossiping solution described in 4.1.2 in such that it also utilizes an unstructured distributed network with a neighbor list. However, instead of using this system to query for (ID, KEY) questions, the nodes in the neighbor list work as the key generators for a *Distributed Identity Based Encryption* system as well as each node keeping track the (ID, KEY) pair of all its neighbors, where the KEY in this case is the public-key for the node ID in the public-key encryption system used for the main part of the InnerCircle protocol.

The key-generation is performed as follows: Whenever a node has joined the network and been allotted its neighbors it sends a key-generation request to each of its neighboring nodes containing (ID, KEY). Each of these neighbor nodes *n* then check their list to confirm that the (ID, KEY) pair is valid before continuing with the DKG to generate its share s_i of the master-key *s*. After that is done, the node *n* responds with their key share d_i and their share of the system public-key s_nP. This response is encrypted using the other public-key crypto scheme to make sure that only the original requester for the key-generation can use the keys hare to create the private-key. The reason for also sending the public-key share is because normally in a distributed IBE setting the system parameters, including the public-key, are global knowledge. However, in the type of setting suggested here this system public-key will be almost unique since every single node will have almost a unique set of key-generators, their neighbors, they are not shared with anyone else. Thus the system public-key *P* will only be usable for encryption regarding the node *n* with ID *id* that requested the key-generation that created the master-key, and thus public-key. Therefore, the public-key share s_nPi has to be part of the response so that the requesting node *n* can form the public-key and append it to all location proximity requests so that the responder can encrypt using IBE. Note that this is a bit different from the previously discussed behavior of a distributed key-generation system as in the *setup* and the *extract phases* are combined into one, which are both executed at the request of a node wanting to have a key-generated.

Important to note here is that the security threshold *t* mentioned during the explanation of distributed IBE has to be set so that $2t + 1$ is slightly lower than the max number of neighbours. This is to make sure that all nodes have a high probability of receiving at least $t + 1$ shares and thus be able to construct their private key *d*.

When a node *n* wants to initiate location proximity with a node *m* it sends a request message of the form {ID, PUBKEY, SYSPUBKEY, ENCRYPTED LOCATION VALUES}, where SYSPUBKEY is the public-key for the node *n* key-generation system. When a request of that form is received the node *m* perform the InnerCircle protocol as described earlier in the thesis.

After a result is produced from the Euclidean distance formula and distance test, this result is encrypted once more using the *Boneh Franklin Identity Based Encryption(BF-IBE)* [9] and then sent back to the node *n*. Note here that the only difference to the original InnerCircle protocol is the extra encryption at the end and how the first request message is constructed.

See figures 21 and 22 below for an illustrative example.

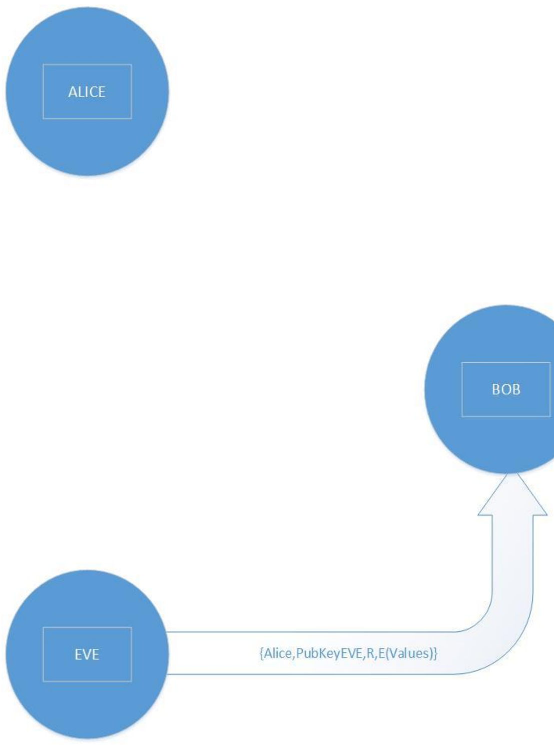


Figure 21. Step 1. Setup is the same as for the example for the Gossip solution with the focus on the three clients Alice, Bob and Eve and with Eve sending a location proximity request to Bob pretending to be Alice. The difference here however is that we forgo the inclusion of their neighbors as they are not needed for the location proximity request handling.

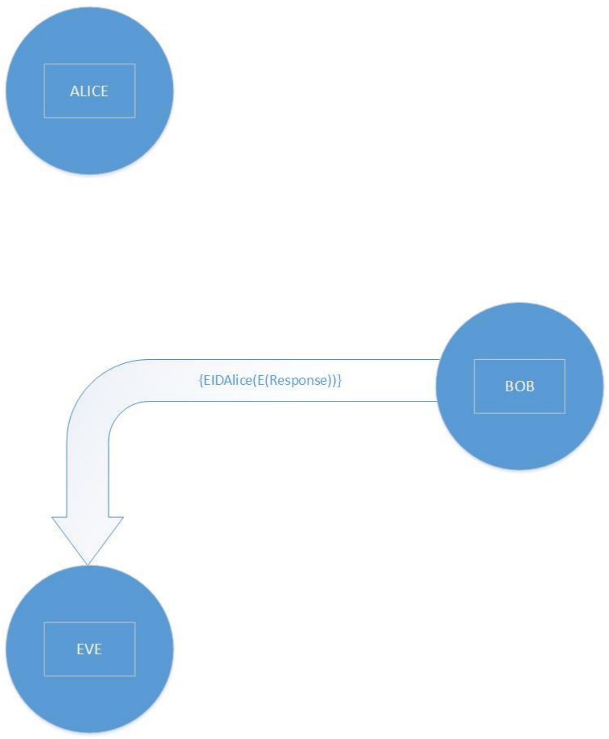


Figure 22. Step 2. After performing the needed calculations Bob responds to Eve with the list of encrypted numbers possibly containing an encrypted 0. However, the entire list has been encrypted an

extra time using the Id Alice as key, thus Eve has no use of this response. Note the different response here compared to the rejection done in the Gossip solution.

4.2.3 What problems does it solve?

As with the corresponding section, 4.1.3, for the gossiping solution, we will here focus on how this solution does in relation to the first two challenges, confidentiality and scaling and showing why it solves the challenges in the security consideration part. The impact on InnerCircle is left for the Testing and Results chapter.

4.2.3.1 Confidentiality Challenge

This is solved since even if a node *n* would pretend to be some other node *m*, the result is encrypted using *m's* unique ID and thus *n* cannot access the result.

4.2.3.2 Storage space scaling Challenge

Just as described for the gossiping solution in section 4.1.3, the storage space scaling challenge is solved since the storage space needed is a fixed value corresponding to the max number of neighbors each node can have. Each node only needs to store (ID, KEY) pairs for its neighbors and not for all the nodes in the entire system. Thus the needed space to store those (ID, KEY) pairs do not grow together with the growth of the network.

However, this solution also worsens the network scaling mentioned in section 3.1.2, since it has to include an extra public-key, the one corresponding to the key-generation system for the requesting node. This makes the messages even bigger and thus scales even worse. However, this does not scale as bad as the gossiping solution in this regard.

4.2.4 The changes to InnerCircle for the solution

As mentioned previously the only changes to the InnerCircle protocol is firstly, that the request message is constructed as {ID, PUBKEY, SYSPUBKEY, ENCRYPTED LOCATION VALUES} instead of simply {PUBKEY, ENCRYPTED LOCATION VALUES} and secondly, that the response to the requester is encrypted using BF-IBE after the main part of the InnerCircle protocol, the distance calculation and test, is completed.

4.2.5 Possible Implementation

The implementation suggestion for creating the neighbor list and the unstructured distributed network is the same as the one described in section 4.1.5 for the gossiping solution. The only difference is that instead of repeating neighbor request until the joining node has either m neighbors or is a neighbor with all nodes that had a free slot for a new neighbor, the new joining node repeats contacting the connection point for a new neighbor until it is either neighbor with all possible nodes in the system or it has reached an amount of neighbors equal to the threshold, $2t + 1$, that was established in the earlier presentation of the distributed version of BF-IBE [10] in section 2.5.2.

4.2.6 Security Considerations

This discussion will be different from the earlier discussion for the gossiping solution in section 4.1.6, since all of the concepts, BF-IBE, distributed key generation for IBE and the suggested construction implementation have all been proved before. Either by other authors in their papers, such as is the case for BF-IBE [9] and distributed key generation for IBE [10], or as part of the discussion for the gossiping solution in section 4.1.6. Therefore, this discussion will focus more on discussing that the double encryption does not compromise the security of the protocol and the slight changes to BF-IBE that would have to be made to make it work to encrypt already encrypted data.

One important initial fact we can start with is that the value of $x^2$and y^2 do not exceed the value of the prime *p* used to construct the prime field *F* over the elliptic curves that are created and thus in which all the cyclic groups *G* operate. This is because, as has been mentioned previously in section 2.3, the prime field *F* is the set resulting from the function $f(n) = n \, mod \, p$. Thus if the squared value of either the x-position or the y-position will be mapped to the same element in a cyclic group *G* as some other value *m*, where $m = x^2 \, mod \, p$ in the case of the x-position or $m = y^2 \, mod \, p$ for the y-position. This is due to the cyclic nature of both the group *G* in question and of the function $f(n) = n \, mod \, p$ and could very well lead to false results due to the calculations being carried out with other values than those intended.

In the same vein as the above paragraph, the elliptic curve used for the second encryption must be larger, or equal, to in size as the elliptic curve used for the first encryption. What is meant by this, is that, if for example the curve used for an Elliptic Curve ElGamal encryption for the first homomorphic encryption scheme is a 256-bit curve, then the curve used for the second encryption using BF-IBE has to be at least a 256-bit curve as well. This is because when discussing elliptic curves as 256-bit curve or 192-bit, that bit size is actually the bit size of the prime *p* of the prime field. Thus if the ElGamal encryption uses a curve over a field with a prime *p* of bit size 256 and the BF-IBE is implemented with curve over a field where the size of *p* is 192-bits, the same problem arises as above, with data being mapped to values that are remapped to some other data, due to the cyclic nature of the fields and groups involved.

There is one very important key difference between this solution and the gossiping one, this solution always returns a result to the requester. However, for this solution that is fine, since the aim is not to stop any malicious clients from the getting the results, only to make it useless for them. Because the second encryption encrypts the data using IBE, any malicious client pretending to be someone else will not be able to quickly decrypt the first layer of encryption using the decryption key belonging to the id they would be faking in their location proximity request, since they simply do not possess the key. Since that attacker lacks the needed key they would have to rely on some type of brute-force attack instead to decrypt the message, something which takes much longer than the normal decryption. However, it does not matter that the malicious client eventually breaks the first layer of encryption and accesses the second layer encrypted using their public key, since breaking the first layer takes enough time for the location proximity result to no longer be reliable. There is a high probability that the receiver of the location proximity request is far removed from the location they were in when the protocol was carried out by the time the first encryption layer is broken through.

However, for the above to be true, there must be no way for a malicious client to generate a key for some id i_o other than their own true id i_t. We shall prove that this is the case for this solution.

Theorem 2: A malicious client with id i_t cannot generate a private key for some other id i_o belonging to another client.

Proof: There are two different possible scenarios, the first being that the malicious client, we will call it Eve, knows which clients are the neighbors to the client whose id i_o it wants to generate a key using, we will call this target client Alice. In the second scenario, Eve has no idea which clients are neighbors to Alice.

For the first scenario, assume that Eve knows all of Alice's neighbors, and thus key generators, and that Eve has managed to extract a key for Alice's id i_o. For Eve to be able to extract the key, she will have to have received at least $t + 1$ key shares from the $2t + 1$ neighbors after sending them a share generation request containing the id i_o and her public key. However, whenever client *c* receives such a share generation request, before generating its share s_c, it checks its list of neighbors for the id in the message, to confirm that it truly is a key generator for the id in the request. After it has confirmed itself as a key generator for the id contained in the share generation request, the client *c* also checks that the public key contained in said request belongs with the id, since, just as in the gossiping solution, it keeps a {ID, KEY} list for all its neighbors. Thus when each honest neighbor *h* of Alice receives the share generation request from Eve, it will confirm that the key supplied by Eve does not match with the id i_o and thus it will not generate its share s_h. Since we assume a majority honest system, Eve will not receive $t + 1$ share and can thus not extract a key for id i_o. The opening assumption for scenario one is thus contradicted.

As previously established, in the second scenario Eve does not know the neighbors of Alice and thus does not know which clients function as key generators for Alice, since these are one at the same. If Eve does not know the key generators for Alice, she does not know which clients to send her key generation request for Alice's id i_o and thus cannot generate a key for this id as she will not be able to contact any client for a key share for that id. This is the most realistic scenario, even if Eve is a key generator for Alice, she would not know which other clients are key generators as they do not need to collaborate at all in the BF-IBE setting [10].

Another consideration to be made is if the double encryption conflicts with itself. However, the principle of the double encryption is the same as the principle behind onion routing [24], which is layered encryption to provide not only confidentiality of the information during the routing but also anonymity since only the edge nodes on the routes will be able to process a unencrypted package [24]. Even if this was not the case, the calculations for the two different encryptions are carried out in different groups, G_1 for the Elliptic Curve ElGamal and G_2 for BF-IBE, meaning that the second encryption will not reveal anything about the first one, since as for the BF-IBE the results of the first encryption is no different from the results of the H_1 hash mentioned in section 2 in the theory chapter, covering BF-IBE. Furthermore, if there were too remain concerns regarding the double encryption the H_1 hash can be modified to be a hash from the group G_1, used for the first encryption, to a third group G_3 which will then be used for the pairing in the BF-IBE encryption.

5. Testing and Conclusion

In this chapter the reader will be provided with information of how the testing and data gathering was carried out and the result of the testing, divided up into three categories: Baseline, Cryptographic Solution and Gossiping Solution, in the form of tables and graphs. The reader will also be presented with a small conclusion and an interpretation of the data leading to an answer of the main research question.

5.1 Testing

5.1.1 Testing Method

The testing was carried out by developing an application using the C# programming language that could do the following:

- Encrypt and Decrypt using Elliptic Curve ElGamal
- Perform the InnerCircle distance calculation and check
- Send messages between at least two devices

This application was used in the following manner to obtain the performance data. Firstly, to obtain a baseline performance, that of simply the InnerCircle protocol, a method is fed with encrypted position values, x, y, x^2 and y^2, of a point A, the unencrypted position values of another point B and an unencrypted distance value. The different position values are then used to calculate Euclidean distance between the two points A and B, as described earlier in section 2.2.4 of the theory chapter in this thesis, discussing InnerCircle. The distance check described in section 2.2.4 is then carried out to create a list that will contain an encrypted zero value if the Euclidean distance is less than or equal to the supplied distance squared, or only random encrypted values otherwise. This list is then decrypted and the decrypted list is checked if it contains a zero and the time it took for the entire process to be carried out is printed to a console window. Secondly, to obtain the performance data for the Cryptographic solution the same approach as for the baseline data is taken, except for a few crucial changes. The list returned from the distance test is encrypted an extra time and is then decrypted twice before being checked for a zero value. Finally, for the data concerning the Gossiping solution again the same procedure as for the baseline data is performed. However, with the slight twist that before the calculations are carried messages are passed between two devices *k* times to simulate how the depth of a network impairs the performance of the protocol. This testing simulates some client with position of the point B receiving a Location Proximity request from some other client with position of the point A.

The reason for double encrypting with Elliptic Curve ElGamal instead of once with Elliptic Curve ElGamal, as normal, and then with BF-IBE as described in section 4.2.2, discussing the Cryptographic solution, in the Solution chapter is twofold. Firstly, time constraints, BF-IBE proved to be very difficult to implement and would have severely increased the time needed for the project. Secondly, as mentioned earlier in this thesis in section 2.5.1, Elliptic Curve ElGamal and BF-IBE has the same performance [10] [19], and thus the impact on the base InnerCircle protocol is roughly the same no matter which encryption scheme is used when only interested in gathering performance data, as we are.

When it comes to the Gossiping solution performance test, the focus is only on how the depth of the network, i.e. the maximum distance between a neighbor of the requester and the receiver of the request. This is because this depth is the highest amount of forwards that will be needed to have reached all nodes at least once and thus dictates the amount of time the receiver of location proximity request will have to wait until having received responses from all of the claimed requesters neighbors.

5.1.2 Results

The performance has been tested using the distance values of 5, 10, 15, 20 and 25 for all three systems, InnerCircle basic, this thesis Cryptographic solution and the Gossiping solution. This is because due to how the response list is created doing the distance test, the distance to test against dictates the execution time of the InnerCircle protocol that is used for all three systems. The Gossiping solution was also performance tested with depths of 20, 200 and 2000 for each of the distance values, as this system's performance is also dictated by the depth of the network.

The performance results are presented below, with the times being rounded to the closest hundred.

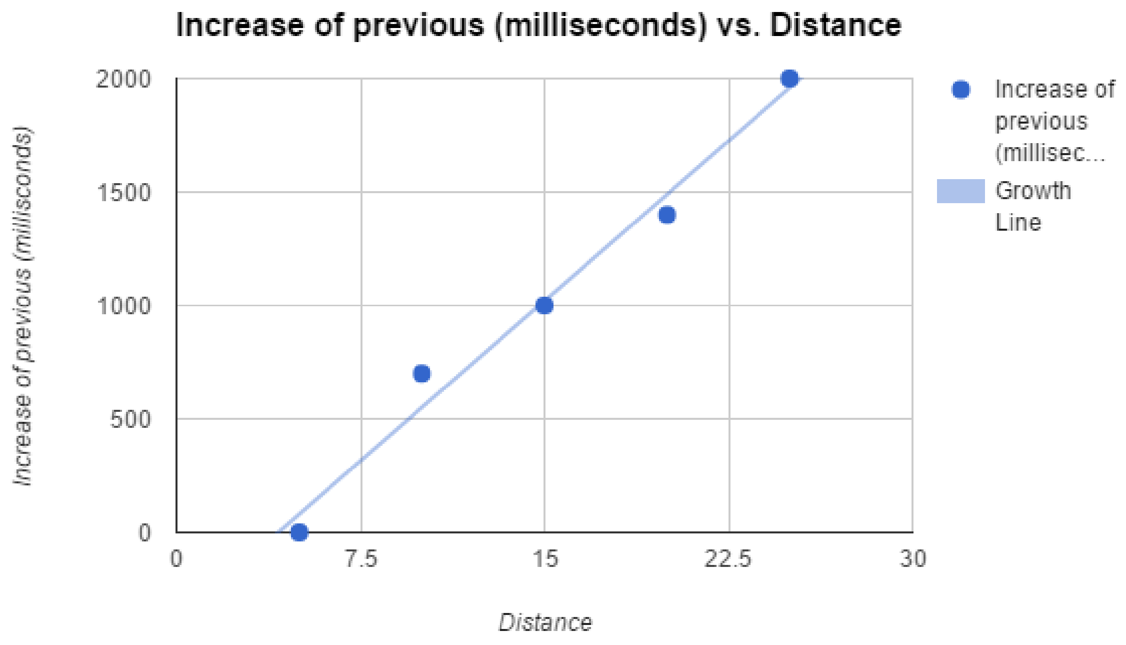
5.1.2.1 InnerCircle

In the following table 5.1 it is shown how the basic distance calculation scales with the test distance, with the focus being on how much each increment of distance affects the performance compared to the previous distance.

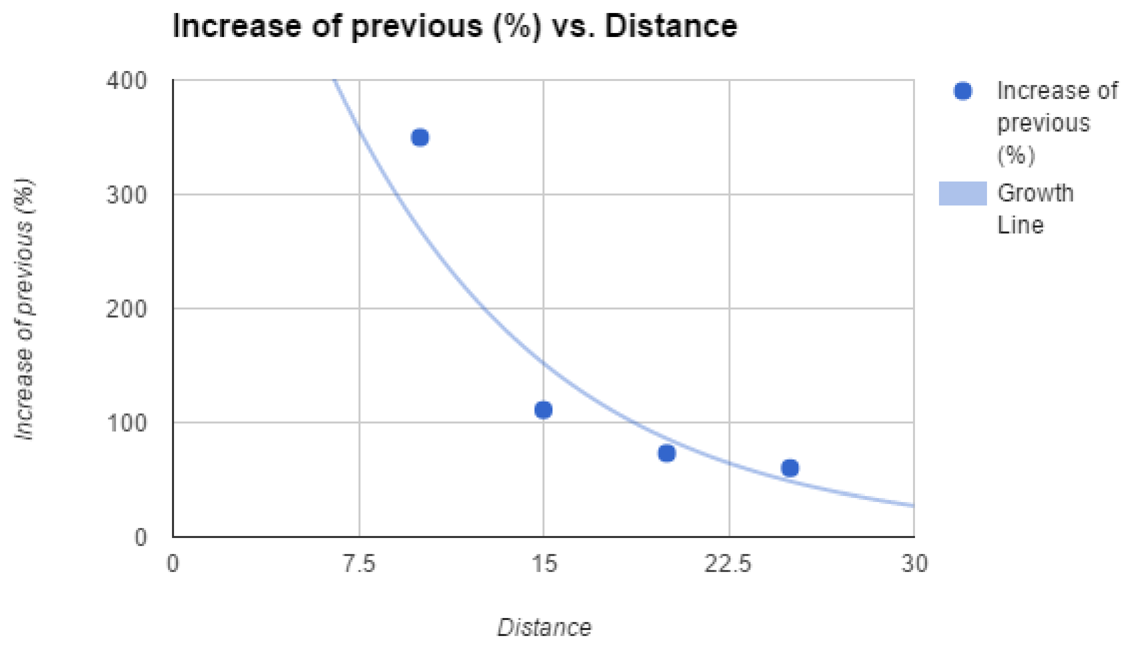
Table 5.1

Distance	Performance(milliseconds)	Increase of previous(milliseconds)	Increase of previous(%)
5	200	N/A	N/A
10	900	700	350
15	1900	1000	111
20	3300	1400	73
25	5300	2000	60

From the above table 5.1 we can see that the execution time of the baseline, InnerCircle protocol increases with larger and larger increments as the location proximity distance grows. However, the percentual growth decreases. The growth graphs are thusly expected to look as follows in Graph 1 and Graph 2:

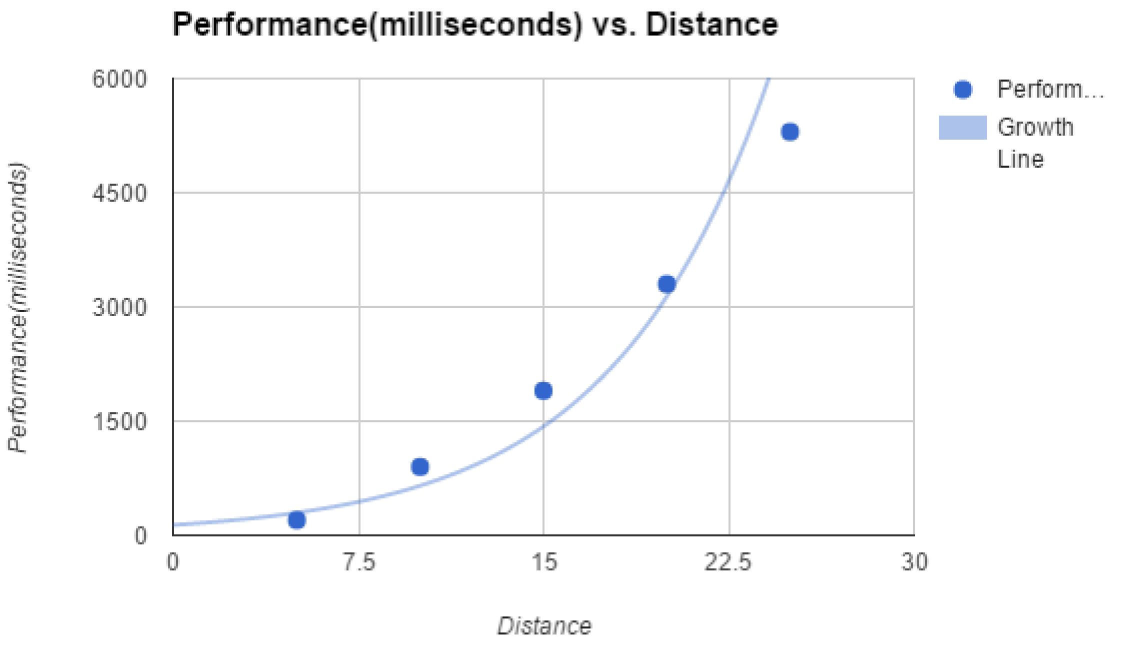


Graph 1. Shows the linear growth of the execution time with regards to distance



Graph 2. Shows the exponential decline of the percentual increase of the previous value in execution time with regards to distance.

Using the above graphs, Graph 1 and Graph 2, and table 5.1 it comes quite natural to reach the conclusion that the base protocol has an exponential growth, as displayed by the following graph, Graph 3.



Graph 3. Shows the exponential growth of the execution time in regards to the distance.

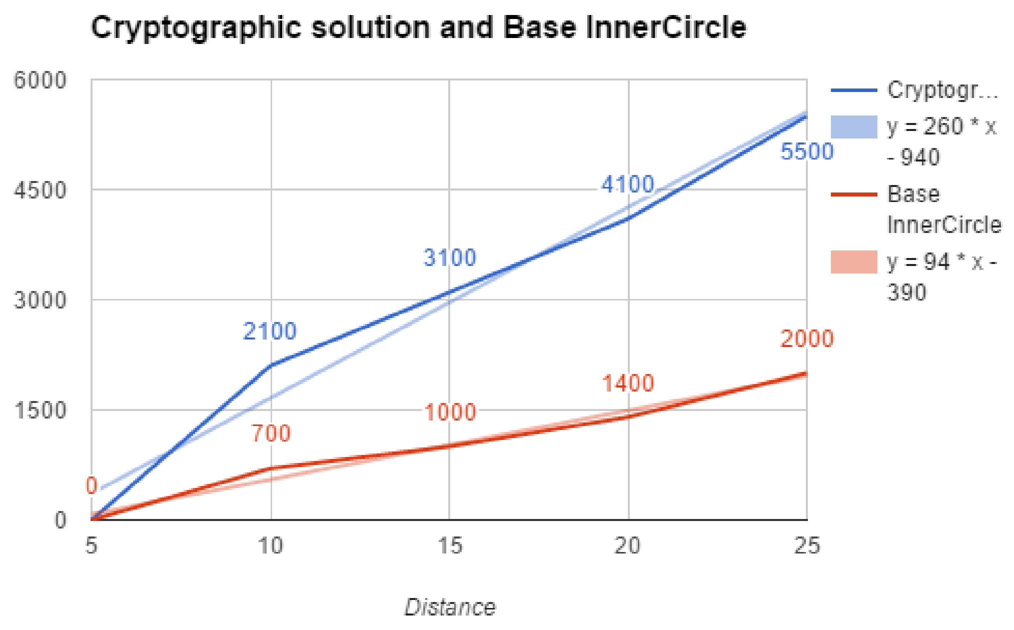
5.1.2.2 Cryptographic solution (Double encryption)

In this section table 5.2 show how the double encryption affects the performance of the baseline protocol. This means that the performance of the protocol using double encryption for each distance will be measured against how the baseline without double encryption performed on each distance. The same will be true for the tables in the later section covering the data from the Gossiping solution.

Table 5.2

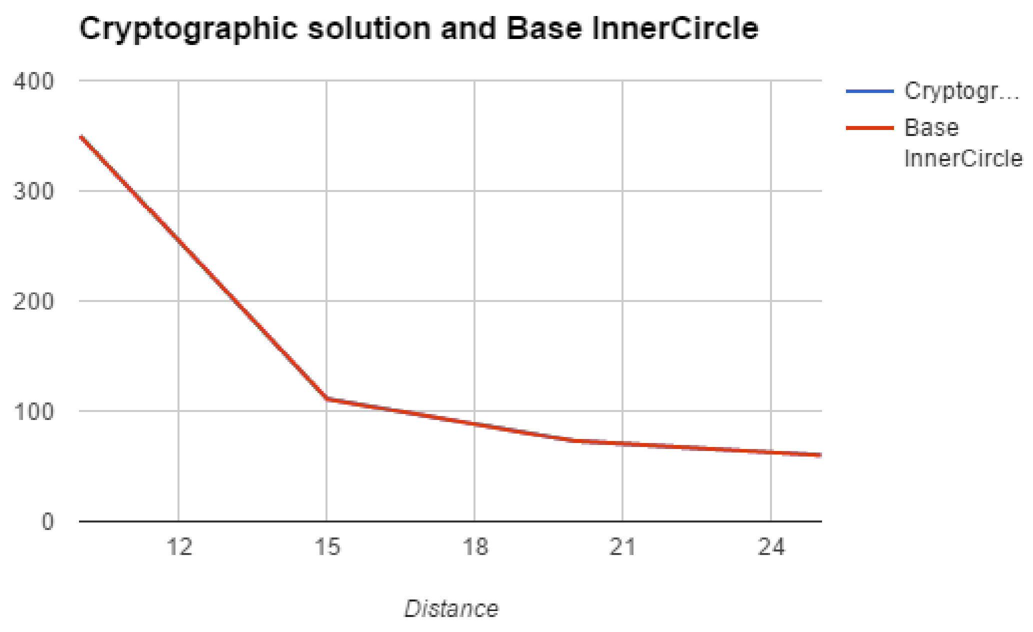
Distance	Performance(milliseconds)	Increase of baseline(milliseconds)	Increase of baseline(%)
5	600	400	200
10	2700	1800	200
15	5800	3900	205
20	9900	6600	200
25	15400	10100	190

From the above table 5.2 showing the performance of the Cryptographic solution with the double encryption scheme as compared to the baseline of normal InnerCircle, we can see that just by adding the double encryption, and not taking into account any additional performance changes from the communication implementation to send the messages, the performance is decreased by roughly by 200% percent across the board, which is an increase of a factor of 3 in the execution time. This means that when the execution time growth of increasing the distance is measured in milliseconds, it will grow faster than the corresponding baseline graph as shown below in Graph 4.



Graph 4. As can be seen, both growth lines are linear, the simple difference being that the line for the Cryptographic solution is steeper.

However, when the growth is measured by percentage instead, it stays exactly the same, as can be seen in Graph 5 below.



Graph 5. When the growth regards to distance is measure by percentage it is exactly the same for both the Cryptographic solution and the baseline.

This is just as expected when there is, as previously mentioned, a flat increase of 200% of the execution time.

5.1.2.3 Gossiping solution

The performance data tables for the gossiping solution will be divided up in five, one for each distance, this because we are focused on how the depth affects the performance. For each table the data for depth 0 is taken from the baseline InnerCircle performance table, this baseline is what all the depths will be compared to.

Table 5.3

Distance	Depth	Performance(milliseconds)	Increase of baseline(milliseconds)	Increase of baseline(%)
5	0	200	N/A	N/A
5	20	800	600	300
5	200	2500	2300	1150
5	2000	16900	16700	8350

Table 5.3 illustrates how the performance of the protocol changes with an increase in network depth at a distance value of 5 and compares the performances of the different depths with the baseline, which is a depth of 0.

Table 5.4

Distance	Depth	Performance(milliseconds)	Increase of baseline(milliseconds)	Increase of baseline(%)
10	0	900	N/A	N/A
10	20	2000	1100	122
10	200	3800	2900	322
10	2000	17800	16900	1878

Table 5.4 does the same as table 5.3 did for distance value of 5, except with a distance value of 10 instead.

Table 5.5

Distance	Depth	Performance(milliseconds)	Increase of baseline(milliseconds)	Increase of baseline(%)
15	0	1900	N/A	N/A
15	20	4000	2100	111
15	200	6200	4300	226
15	2000	21200	19300	1016

Table 5.5 does the same as table 5.3 did for distance value of 5, except with a distance value of 15 instead.

Table 5.6

Distance	Depth	Performance(milliseconds)	Increase of baseline(milliseconds)	Increase of baseline(%)
20	0	3300	N/A	N/A
20	20	6900	3600	109
20	200	8400	5100	155
20	2000	22800	19500	591

Table 5.6 does the same as table 5.3 did for distance value of 5, except with a distance value of 20 instead.

Table 5.7

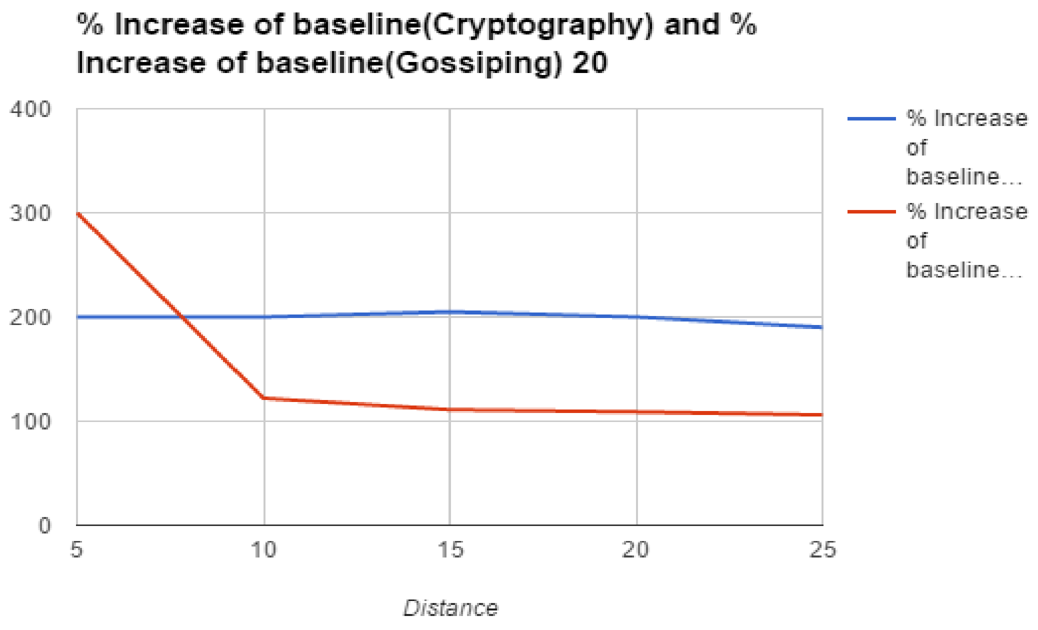
Distance	Depth	Performance(milliseconds)	Increase of baseline(milliseconds)	Increase of baseline(%)
25	0	5300	N/A	N/A
25	20	10900	5600	106
25	200	13700	8400	158
25	2000	25600	20300	383

Table 5.7 does the same as table 5.3 did for distance value of 5, except with a distance value of 25 instead.

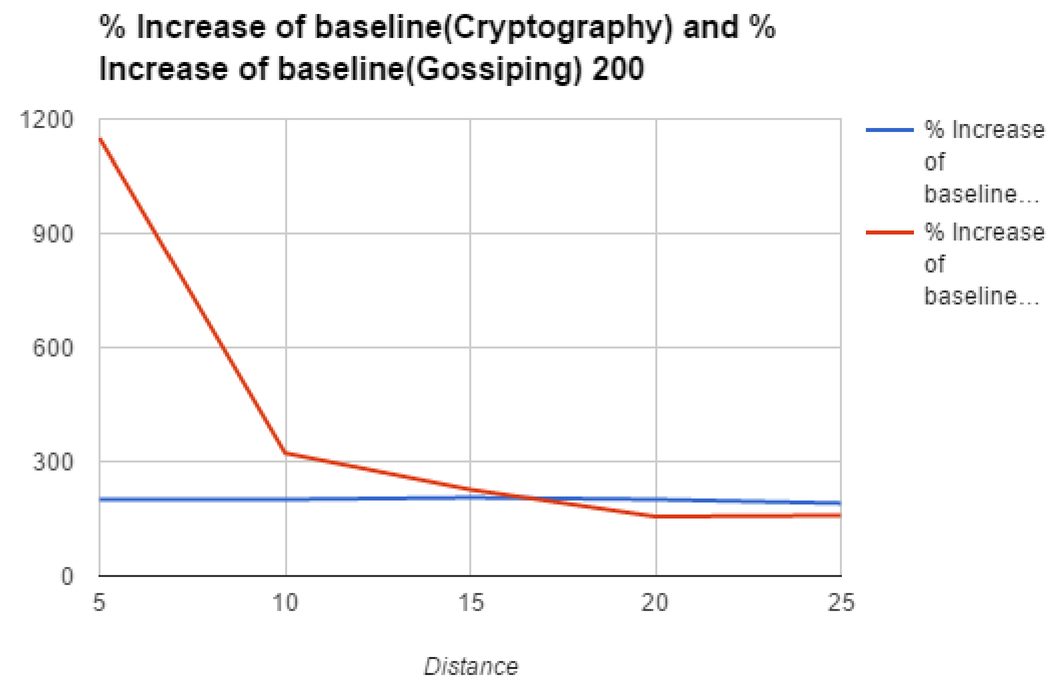
While there are a lot of numbers in the tables 5.3 to 5.7 above, one thing becomes clear quite early and that is that as the distance grows larger the Gossiping solution performs better than the Cryptographic solution. As can be seen by comparing the tables in section 5.1.2.2 and this section is that as the distance values grows the impact of the depth of the network in the Gossiping solution becomes less and less of a factor and appears to stabilize at around a doubling of the performance of the base InnerCircle protocol. The Gossiping solution already performs better at such low distance values as 20 with a network depth of up to 200. If this distance was measured in meters then that means that in the scenario of two people being close enough the see each other the Gossiping solution already performs better than the Cryptographic solution at a network depth of 200, as in even if the receiver of the location approximation request needs to wait for their authentication request to forward 199 times after having it sent to it to all its neighbors it would still finish before the Cryptographic solution.

However, the above tables, 5.3 – 5.7, for the Gossiping solution does not take into account the time it takes for each node to do a lookup for the ID attached in the authentication question sent by the location proximity receiver. Depending on how long this lookup takes it could highly increase the execution time for the Gossiping solution, the extra time added would roughly be the lookup time times the depth of network. However, this will eventually outperform the Cryptographic solution regardless, it would just require higher distance values before it happens.

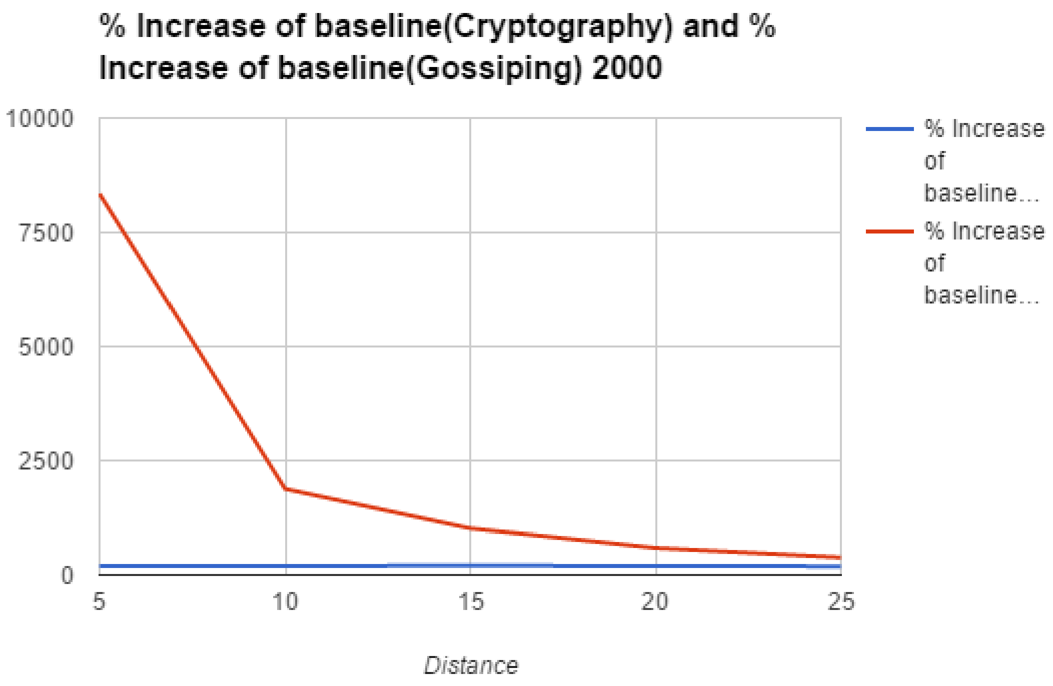
Following is graphs, graphs 6,7 and 8, that compares the percentual increase of execution time of the Cryptographic solution, which is flat as mentioned earlier, with the percentual increase of the execution time for all three tested depths of the Gossiping solution. The increase discussed here is how much each solution increases the execution time of the baseline location proximity protocol, InnerCircle.



Graph 6. The blue line represents the percentual increase with the Cryptographic solution and the red line represents the same for the Gossiping solution with a depth of 20.



Graph 7. The blue line represents the percentual increase with the Cryptographic solution and the red line represents the same for the Gossiping solution with a depth of 200.



Graph 8. The blue line represents the percentual increase with the Cryptographic solution and the red line represents the same for the Gossiping solution with a depth of 2000.

Just as discussed, with a low depth the Gossiping solution starts outperforming the Cryptographic solution almost instantly. Furthermore, the same is true even for a medium depth and even for high levels of depth the performance of the Gossiping solution catches up with the performance of the Cryptographic solution, and passes it, even for quite short distances.

5.2 Conclusion

We touched on the conclusion in the previous results section, 5.1.2, and that is that the Gossiping solution will eventually outperform the Cryptographic solution no matter the network depth as the distance grows. The reason for this is that the Gossiping solution is not as dependent on the distance as the Cryptographic solution, since as the query distance grows so does the size of the list the Cryptographic solution needs to encrypt and decrypt an extra time. This fact leads to the flat increase in execution time for the Cryptographic solution compared to the baseline.

Another thing to mention is that the percentual increase on the execution time from the baseline in the Gossiping solution will actually trend towards 0% as the distance grows. The reason for this comes from how the InnerCircle protocol does the distance test, that it subtracts every single value between 0 and d^2 from the Euclidean distance [1]. This fact means that the number of subtractions, and thus elements to check if for a zero value, increases exponentially as the distance *d* increases. Thus the answer to the initial research question for this thesis, if it is possible to add built authentication to a decentralized privacy-preserving location proximity protocol without severely decreasing the performance, is an emphatic *yes*. However, this solution will severely increase the network scaling issues already present in the base protocol, as previously explained in section 4.1 covering the Gossiping solution in the Solution chapter.

6. Future

In this chapter possible future work in the location proximity subject that could possible improve on the solutions put forward in this thesis are presented. I also provide a base to build upon for this possible future work.

6.1 Future Work

For future work I would recommend developing an Identity based cryptography system that is additively homomorphic. That type of cryptosystem would solve the authentication problem without worsening the network scalability problem to the extent the Gossiping solution I have put forward does. If this hypothetical cryptosystem also has a performance that at the very least matches Elliptic Curve ElGamal it would perform just as well as the InnerCircle protocol used as a baseline in this thesis.

To achieve the above I would recommend starting with the Basic Implementation put forward in the BF-IBE article [9], since it is my belief that if the hash $H_2: G_2 \to \{0,1\}^n$ is changed instead to $H_2: \{0,1\}^n \to G_2$ and since for the location proximity only ever handles positive real numbers, this can be changed further to $H_2: Z^+ \to G_2$. With these changes the ciphertext can be changed from $C = (rP, M \oplus H_2(g_{ID}^r))$ to $C = (rP, H_2(M) + g_{ID}^r)$ and the decryption from $M = (M \oplus H_2(g_{ID}^r)) \oplus H_2(e(d_{ID}, rP))$ to $M = (H_2(M) - g_{ID}^r) - e(d_{ID}, rP)$ with the addition and subtraction functions being those defined as group operations. With these changes it is my belief that the Basic Ident scheme in BF-IBE becomes additively homomorphic and will work in such away that it becomes conductive to use for location proximity as it has been described in this thesis. However, due to time constraints I have not been able to prove that this conceptualization of the Basic Scheme in BF-IBE makes it homomorphic, neither have I been able to construct the security considerations to prove that this version is still secure. Also thanks to the fact that the BF-IBE scheme can have distributed key-generation [10], any Location Proximity protocols that implements it would have the possibility to be decentralized, as desired in InnerCircle for example [1].

7. Ethical and Social considerations

Here the thesis presents a possibility of how a fully practical location proximity application could be used to impact the everyday life of average people through the use of location based advertising. Two different ways of looking at the possibility of location based advertising and how people might feel about it are also presented in a short conclusion.

7.1 Possible social impact of location proximity

The main ethical and social implications regarding this thesis I can think of has more to do with general development in the field of location proximity than it has with this actual thesis. The implication I am thinking of is the possibility of even more directed advertisement through, or with the help of, location proximity applications. An example of this would be companies paying the developer of a location proximity application, or developing one themselves, for the privilege to be able to query, and having their queries accepted, by all other clients using the same application. In this scenario a company could repeatedly query the clients of ordinary users using some distance *d* and when a client falls within the distance *d*, target that client with advertisement and personalized coupons such as: “**Hello! You have been randomly selected to receive an extra donut with every donut you buy for FREE! This offer can be utilized at any Fred's Donut Shop expires in 10 minutes**”. The company could even add in directions to the closest retailer, which in this case would be the retailer that queried the client and found them to be within distance *d*. In this same scenario the location proximity application can be used to establish patterns for different clients for a company to exploit. Let's say for example that the client belonging to the user Alice comes within in distance *d* of Fred’s Donut Shop at roughly the same time twice a day multiple days a week. Using that information, it is quite simple to draw the conclusion that Alice passes by within the distance *d* of shop to and from her way to work. That type of information and analysis could be used to try and time advertisement so it is sent to client before they assume their travels, perhaps even in conjunction with big data analysis to customize the adverts after what people are most inclined to purchase on their way to and from work, to increase the probability of a user of the location proximity application becoming a customer of the company.

The above coupled with how today's advertising business already utilize big data analysis, as in to use customers shopping behavior to predict what they could be in need of in the near future to market specific items to the same customers and tracking and gathering information online to create more personalized online advertisement, could make advertisements even more effective. More effective advertisement can be seen as both good and bad, having adverts customized for each person means that people will only be offered deals for and subject to ads regarding services when there is a high probability that they will have some interest in. Together with the possibility of location based advertising by using location proximity would also mean that these adverts would only be from retailers and services within a close distance *d* of the person in question. However, the price to pay for the convenience of only being advertised services one might have use of, is that some faceless corporation has access to enough information about the person in question to be able to facilitate this. This means that the corporation behind the target advertisement knows almost everything about a person, their interest and hobbies, their relationship status and their behavior regarding their economy, since this information can be gathered through a person's online browsing habits and their consumption pattern. Furthermore, in the instance presented in the previous paragraph regarding the use of location proximity for advertisement, using the location of the store in question and the timing of a client becoming within distance *d* can give the same corporation a rough guess of where that person works and where they live.

With the discussion in the previous two paragraphs in mind a conclusion such as follows can be drawn. Even if the aim of location proximity, which is the field in which this thesis operates and aims to contribute to, is to provide users with the possibility of location based services whilst still preserving their privacy it has the potential of having the opposite effect of providing big corporations, and perhaps malicious parties if those same corporations handle the security of their data poorly, even more of our personal information and daily routine. However, judging by the popularity of the recently released mobile game Pokémon Go, most of the population does not seem to mind handing over the data of their exact location and their transport patterns in exchange for the utilization of an attractive enough product.

References

[1] Hallgren, P., Ochoa, M., & Sabelfeld, A. (2015, July). Innercircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *Privacy, Security and Trust (PST), 2015 13th Annual Conference on* (pp. 1-6). IEEE.

[2] Location based services startups. (n.d.). Retrieved from https://angel. co/location-based-services.

[3] Coldewey, D. (2012). Girls Around Me. *Creeper App Just Might Get People To Pay Attention To Privacy Settings. TechCrunch*.

[4] Phillips, A., Schroth, F., Palmer, G. M., Zielinski, S. G., Smith, A. P., & Cunningham III, C. M. (2010). *U.S. Patent No. 7,848,765*. Washington, DC: U.S. Patent and Trademark Office.

[5] Li, X. Y., & Jung, T. (2013, April). Search me if you can: privacy-preserving location query service. In *INFOCOM, 2013 Proceedings IEEE* (pp. 2760-2768). IEEE.

[6] Costella, C. *Pairings for Beginners* [PDF document]. Retrieved from http://www.craigcostello.com.au/pairings/PairingsForBeginners.pdf

[7] Wolframalpha. (n.d.) https://www.wolframalpha.com/input/?i=y%5E2%3Dx%5E3+-+x%2B1.

[8] Shamir, A. (1984, August). Identity-based cryptosystems and signature schemes. In *Workshop on the Theory and Application of Cryptographic Techniques* (pp. 47-53). Springer Berlin Heidelberg.

[9] Boneh, D., & Franklin, M. (2001, August). Identity-based encryption from the Weil pairing. In *Annual International Cryptology Conference* (pp. 213-229). Springer Berlin Heidelberg.

[10] Kate, A., & Goldberg, I. (2010, September). Distributed private-key generators for identity-based cryptography. In *International Conference on Security and Cryptography for Networks* (pp. 436-453). Springer Berlin Heidelberg.

[11] Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems Principles and Paradigms 2nd edition*. Pearson. (pp. 1- 32)

[12] Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems Principles and Paradigms 2nd edition*. Pearson. (pp. 33- 68)

[13] Fraigniaud, P., & Gauron, P. (2003). The content-addressable network D2B.

[14] Yi, X., Paulet, R., & Bertino, E. (2014). *Homomorphic encryption and applications*. Springer. (pp. 27 – 46)

[15] Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, *22*(6), 644-654.

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

[17] Goldreich, O. (1998). Secure multi-party computation. *Manuscript. Preliminary version*, 86-97.

[18] Silverman, J.H (2006). An Introduction to the theory of Elliptic Curves [PDF]. Retrieved from http://www.math.brown.edu/~jhs/Presentations/WyomingEllipticCurve.pdf

[19] Boyen, X. (2008). A tapestry of identity-based encryption: practical frameworks compared. *International Journal of Applied Cryptography*, *1*(1), 3-21.

[20] Kapoor, V., Abraham, V. S., & Singh, R. (2008). Elliptic curve cryptography.*Ubiquity*, *2008*(May), 7.

[21] ElGamal, T. (1984, August). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques* (pp. 10-18). Springer Berlin Heidelberg.

[22] Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, *48*(177), 203-209.

[23] Pedersen, T. P. (1991, August). Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference* (pp. 129-140). Springer Berlin Heidelberg.

[24] Dingledine, R., Mathewson, N., & Syverson, P. (2004). *Tor: The second-generation onion router*. Naval Research Lab Washington DC.

[25] Rouse, M. (2007, June) What is public key certificate? - Definition from WhatIs.com. Retrieved from http://searchsecurity.techtarget.com/definition/public-key-certificate