Realizing and Implementing Location-based Access Control Policies

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik
Computer Science

Realizing and Implementing Location-based Access Control Policies

Master Thesis

Daniela Meier
daniela.meier@alumni.ethz.ch

Wednesday 25th September, 2013

Advisers:
Prof. Dr. Srdjan Čapkun
Elli Androulaki
Claudio Marforio
Claudio Soriente

Department of Computer Science
ETH Zürich

Acknowledgements

I would like to thank my advisors Elli Androulaki, Claudio Marforio and Claudio Soriente for supervising me. I would not have chosen to work on this intriguing topic without Elli who proposed it to me and supervised me for the first two months.
I owe special thanks to Claudio Marforio and Claudio Soriente who wholeheartedly took care of me after Elli left ETH. Claudio Marforio's insight into C and Objective-C has made it possible for me to learn a new programming language in the short time which was available. Without him, I'd probably still be looking for some of the bugs in my implementation. The theoretical part of my thesis was furthered greatly from Claudio Soriente's vast knowledge of cryptography. Moreover, I owe both of them many thanks for their detailed comments when proofreading my thesis.

I am thankful to Prof. Čapkun for providing me with the opportunity to write my thesis in his group. Also entitled to many thanks are his group as well as part of Prof. Basin's group for all the support I received from them.

I would also like to thank Mathias Payer, Andreas Ritter and Rahel Zoller for proofreading my thesis and providing me with invaluable suggestions and discussions.

Special thanks as well go to Andreas, Filip, Lynn and Rahel who patiently provided me with moral support (and daily ice cream).

Abstract

User anonymity is a much-needed and sought-after property of users of internet-based services. With the increased use of internet-based services and cloud-based hosting solutions, users have become more privacy-aware. Unfortunately, recent news has shown that the involved parties do not only receive the intended data but also gain access to other data such as user events or logs. In light of these considerations, it is all the more important to protect information such as data contents and identities of involved parties accessing it as much as possible.

We investigate the topic of user anonymity when accessing cloud-stored files. Access to files is not only granted based on the user's identity, but also based on current time and location of the user. We assume that the cloud provider operates independently of the location assessing infrastructure. We furthermore do not put any extra trust into any part of the system than what is absolutely necessary.

We propose a protocol which allows a user to stay anonymous during file access in the mentioned scenario. We build our solution on top of LoTAC, which provides location and time-based access control. Our protocol makes use of techniques borrowed from blind signatures and zero-knowledge proofs. We show that our protocol does not leak any information about the identity of the user, i.e., is safe against malicious users and also if the location assessing infrastructure is colluding. Finally, we implement LoTAC as an iPad application. We show the feasibility of LoTAC to be used in mobile environments by experiments we conduct on an iPad.

Contents

Chapter 1

Introduction

This chapter will give an introduction to this thesis. We do that by motivating the idea behind the system we developed. In Section 1.2 we will explain the exact problem we solve before giving an overview over the existing system, LoTAC, in Section 1.3. In Section 1.4 we will state our contributions and then conclude with an overview over the rest of this thesis.

1.1 Motivation

In the "old days", where no digital files existed and access control meant having a key to a matching lock, if you wanted to access a file, you had to get to the place where it was stored. If the files were not meant for everyone to see, then you either needed to have the key to the filing cabinet or you needed to show an identification card to the assistant who was in charge of the filing cabinets. In any case, regardless whether your visit was logged or not, you would have to trust the assistant to correctly carry out his duties. As a file owner, you decided where your files may be accessed by deciding where they were stored. You decided who may access them by either handing out keys to the respective people or by handing a list of people to your assistant. In order to ensure access control, you had to trust the people who had a key or trust your assistant to check people's identities correctly.

Today, we have digital files, often stored in the cloud or more generally on remote servers, the new incarnation of yesterday's cabinets. If you do not want them to be accessible from everywhere, you have to take additional measures. Furthermore, if you want to limit who has access to them, you have to implement some kind of access control mechanism.

As a client, if you want to access a file, you may do that either by asking the file owner to send the file to you, or you may fetch a file from a remote host. Most of the time you then need to login to where the file is stored. You need to trust the file storage that the login is handled correctly. Potentially, all of your actions will be monitored.
As a file owner, if you decide to host your file remotely, you need to trust the hosting service to handle access control correctly. (Note that the hosting service takes a similar role as the assistant did once. The difference being, that the file hosting service most of the time is not on your payroll and therefore abides by his own rules in regard of e.g. the data he collects from the users.) Moreover, most of the time it is not feasible to check if the user accessing a file resides at a location you approve of.
If you handle file access manually by letting people ask you for a file and send it to them, you do not need to trust an additional actor, but increased your workload tremendously. Furthermore, the file will not be accessible at all times but only when you are reachable. And if you want to restrict access of a file to certain locations, you most probably need to trust the user to report it correctly.

Obviously, this situation is not ideal. We would even argue that it got worse. A file owner needs to trust a lot of people to achieve his goals without any security breaches. Not only did the number of people to trust increase, the authority you have over them decreased as well. Much more data is collected about users actions, but it did not directly transpose into more security.

1.2 Problem Statement

As we have described, the status quo regarding access control over files is not satisfactory. We favor a setting where no one but the file owner decides who may access a file and where he must trust no one else than himself to correctly identify users.
We want to achieve location-based access control without having to trust the user submitting his location correctly. We instead want a user's location to be assessed by someone capable and trustworthy to do that. On top of that, we do not want to trust the location assessment authority to perform any other tasks than location assessment, especially not access control enforcement based on a users location.
Since we do not have control over the hosting service, we do not want to trust it to perform any critical tasks, be it authentication, access control enforcement or an assessment of a users location.

In addition, we do not want anyone to collect data which does not help to augment security and is therefore not necessary. In other words, we want users to be able to acquire a file anonymously, but still be sure, that the user is indeed allowed to access the file. No hosting service and no location assessment authority should be able to tell whether a certain user has accessed or even only tried to gain access to a certain file.

1.3 Background

We build our solution on top of LoTAC [1]. In LoTAC, a file owner encrypts his files for the users who are allowed to access them and for the locations at which the files may be accessed. He specifies at which location and times the files may be accessed in a so-called “policy” and encloses this policy to the encrypted file.
LoTAC assumes that the owner uploads the files to a hosting service in the cloud (referred to as “cloud storage”). The cloud storage is not expected nor trusted to do any access control enforcement: anyone may download an encrypted file.

To access a file, a user needs to download the encrypted file from the cloud storage. The user is then expected to remove the location-layer of the encryption at the corresponding location. This works by handing over the encrypted file to a so-called “location server” who assesses the location of the user and decrypts the file with the secret key of its location. By doing so it does not care whether the file really was encrypted for this location; it is not the location server’s task to check whether the user complies with the policy.
After the encryption layer for the location has been removed, the user may decrypt the file using his secret key. Clearly, this only works if the file was encrypted for this user.

1.4 Contributions

LoTAC, as it is, supports access control to cloud stored files based on location and time. What LoTAC however can not guarantee is user’s anonymity. Actors of the system can collect data about the users and their actions as they please. In this thesis we show how to change that. We introduce a new feature to LoTAC which enables users to stay anonymous and which prevents location servers to know whom they interact with.

Potential applications include all settings where the location assessment authority does not need or have the ability to identify the users. This may include e.g. most kinds of self-managed networks such as peer-to-peer networks, Bluetooth and sensor networks and certain kinds of WiFi networks as well. The location server infrastructure then consists of servers who are connected to the network in a way which enables them to assess a user's location.

We furthermore implement the existing LoTAC system as an iPad application with owner and user functions. In the application, the owner may choose users who are allowed to decrypt a file and locations and times at which a file may be decrypted, then the file is encrypted and uploaded to a cloud storage. The user is able to choose the file which he wants to decrypt (a list is requested from the cloud storage), and then decrypt the file on the iPad.

1.5 Thesis Structure

The rest of this thesis is organized as follows: First, we explain the background, i.e. LoTAC, the system we build our solution on, in detail. Then, we present our solution, enabling users to maintain anonymity when accessing files, in Chapter 3. Chapter 4 introduces our implementation of LoTAC and Chapter 5 shows the results of its performance evaluation. At last, we talk about related work before we conclude and hint at future directions in Chapter 7.

Chapter 2

LoTAC

Due to its importance to the specific implementations of this project, we summarize the main aspects of LoTAC which is described in [1].

In this chapter we first give an overview of the system in Section 2.1. We then introduce the general architecture in Section 2.2, i.e., the necessary actors along with their tasks, the assumptions made and give a short overview of the protocol. In Section 2.3 we introduce the encryption scheme used by LoTAC before we describe the four phases of the protocol in detail in Section 2.4. At the end, a security argument is given in Section 2.5.

2.1 Overview

The goal of LoTAC is to provide location-based and time-based access control to cloud-stored data by means of encryption. LoTAC assumes a pre-existing location assessing infrastructure and cloud storage which are independently operated systems. It integrates into these existing infrastructures with only small extensions to the location assessing infrastructure and none to the cloud provider. LoTAC does not require any access control to be performed by either the cloud storage or the localization infrastructure. The amount of trust put into each part of the system is minimal. It is assumed that some kind of partitioning of locations exists which renders a location server responsible for a specific location.

The time restriction of the access policy may be implemented as an arbitrary string called “time-tag”. The actors need to agree on a format which is then used by the owner to encrypt and by the user to decrypt. A multitude of possibilities are imaginable, e.g. “January”, “Monday”, “14:00 - 16:00” etc. or a combination thereof. The location server however needs to check whether the string submitted by the user complies with current time. In the system we implemented (see Chapter 4) we used time-tags of the form “hh:mm - hh:mm”.

2.2 Architecture

2.2.1 Actors

LoTAC requires five actors: a file owner, users, the localization infrastructure, location servers and some kind of cloud storage.

The Localization Infrastructure defines the necessary parameters for the encryption and generates location servers’ private and public keys. It also acts as a key server for public keys of users and location servers which are needed by the owner to encrypt files. It is assumed that the location server’s private keys are distributed through a secure channel.

File owners encrypt the file and specify which users may access it and the locations and times where and when the file may be accessed. The file is encrypted twice. The first layer is to be removed by the user, the second one by the location server. The owner then uploads the encrypted file and the access information (which is referred to as the contextual policy) to the cloud storage.

Users download the encrypted file from the cloud storage. When they are at the correct location, they authenticate with the location server and submit the encrypted File. The location server removes a layer of encryption and hands back the file, still encrypted, to the user. The user may then finally decrypt the file to gain access to its contents.

Location Servers interact with the users. They authenticate the users and get their public keys from the localization infrastructure. The user then sends the location server an encrypted file and a time-tag. The location server checks whether the time-tag corresponds to the current time and uses the tag to decrypt the file. After decryption, the location server re-randomizes the file using the public key of the user. It then hands the ciphertext back to the user. The Location Infrastructure is only trusted to authenticate users, not to authorize them in respect to the file they want to decrypt. I.e., it is oblivious to the specified access set of a file.

Cloud Storage saves the encrypted file along with the contextual policy. It allows every user to download regardless of whether the user is allowed to encrypt the respective file. It is only trusted to reliably store the ciphertexts and make them available upon request.

2.2.2 Assumptions

One of LoTAC's strengths is that it requires only minimal trust in its actors. Every actor is only trusted to do his task:

- **The localization infrastructure** sets up the encryption scheme parameters, generates location server keys and is assumed to let the location servers know of their private keys in a secure way.
- **The owner** is trusted to choose a list of users who may access a file and a list of locations/times at which they may access it. Moreover, he is assumed to encrypt the file accordingly and store it to the cloud server.
- **The cloud server** correctly stores the data sent to it and makes it available to anyone requesting a file.
- **The user** is not trusted to do anything correctly.
- **The location server** identifies the user and checks whether the current time corresponds to the time-tag a user sends to it.

LoTAC does explicitly *not* trust:

- **The cloud server** to do any access control.
- **The user** to not try to decrypt files which are not intended for him or at places and times which do not correspond to the policy.
- **The location server** to check whether its or the user's location or the provided time-tag corresponds to the policy or whether the user is allowed to decrypt the file.

2.2.3 Protocol

An overview over the protocol is given in figure 2.1. It consists of three phases: the setup phase which is handled automatically by the localization infrastructure and the location servers, second, the encryption phase whose main actor is the owner of a file and third, the decryption phase where a user tries to decrypt a file after having visited the necessary locations. Below is a more detailed overview over each phase.

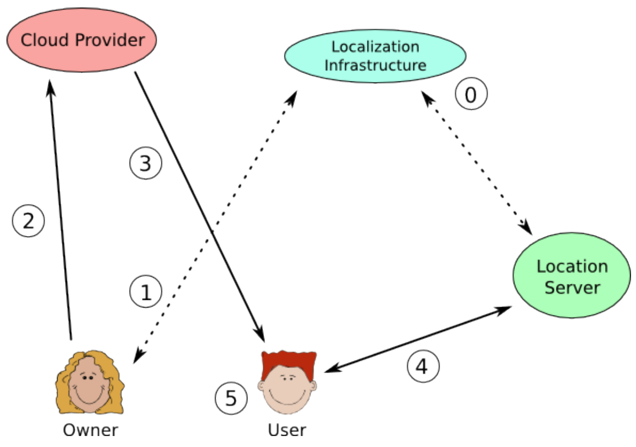


Figure 2.1: Overview of LoTAC protocol phases. 0: Setup phase, 1: Encryption phase, 2: Upload to cloud server, 3: Download from cloud server, 4: Interaction with location server, 5: Decryption

Setup During the setup phase, the localization infrastructure instantiates the parameters of the encryption scheme. Moreover, it generates location servers' public and private keys and builds a list of users' public keys.

Encryption Before encrypting a file, a file owner first defines an access set U_F of users who are able to access the file. He then requests the public keys of these users from the localization infrastructure. Second, he decides when and where the file may be decrypted — this information is called the contextual policy — and requests the necessary public keys of the respective location servers from the localization infrastructure.
It is not until then that the owner may begin to encrypt the file. Encryption is done using a semantically secure encryption cipher and a key K which is generated by the owner. This key K is then further encrypted. First, a user-specific encryption layer is added using the user's public keys. Second, a location-specific encryption layer is added using the location's public keys and the permitted time frame as a tag.
Finally, the resulting ciphertexts are uploaded to the cloud provider for everyone to download.

Decryption When a user wants to decrypt a file, he downloads the corresponding ciphertexts along with the contextual policy from the cloud provider. He then needs to travel to all the locations mentioned in the contextual policy.

Being at the desired location, the user interacts with the location server at that place which decrypts the first layer. If necessary, he additionally visits other locations to let the respective location servers decrypt further layers. After all locations in the contextual policy have been visited, the user may decrypt the remaining layer of encryption using his private key. The resulting plaintext corresponds to the key which he may use to decrypt the file.

2.3 Prerequisites

LoTAC uses ElGamal encryption and a variant of it which we explain in this Section.

2.3.1 ElGamal Encryption

LoTAC encrypts messages as follows: $EG.Enc\left(pk, m, r\right) = \langle m \cdot pk^r, g^r \rangle$ where pk is the public key of the user who is able to decrypt, m is the message and r is chosen randomy $\in \mathbb{Z}_{N/4}$. N and g are public parameters chosen at setup time.
Decryption is defined as $m \leftarrow EG.Dec\left(sk; \langle \alpha, \beta \rangle\right) = \frac{\alpha}{\beta^{sk}}$.

2.3.2 Tag-based ElGamal Encryption

Furthermore, LoTAC makes use of a variant of ElGamal encryption called tag-based ElGamal which is based on tag-based encryption as defined by Mackenzie, Reiter and Yang [20].
Tag-based ElGamal encryption is defined as follows: $TB.Enc\left(pk, \tau; m; r\right) = \langle m \cdot H\left(pk^r, \tau\right), g^r \rangle$ for τ being an arbitrary string, $H\left(\cdot\right)$ a hash function, m, r, pk defined as in ElGamal encryption.
Decryption is defined as $m \leftarrow TB.Dec\left(sk; \langle \alpha, \beta \rangle; \tau\right) = \frac{\alpha}{H\left(\beta^{sk}, \tau\right)}$.

2.4 The Protocol of LoTAC in detail

In this Section we give a detailed description of the protocol we briefly introduced in Section 2.2.3. As before, we distinguish between the setup phase and the encryption phase. Then, we divide the decryption phase into the two Subsections “User-location server interaction” and “Decryption”. At the end, we give a security argument.

2.4.1 Setup

As explained in Section 2.2.3, at the beginning the localization infrastructure instantiates ElGamal in $\mathbb{QR}_N^+$. It does so by choosing a group $\mathbb{G}$ with generator g and publishing the modulus N. $\mathbb{G}$ is defined as the group of signed quadratic residues $\mathbb{QR}_N^+$ (see [16] for more details about signed quadratic residues). N is chosen as $p \cdot q$ where p and q are safe primes, i.e. of the form $2p' + 1$ where p' is prime as well.
Moreover, the localization infrastructure generates location servers' public and private keys and builds a list of users' public keys to be accessed by the owner and location servers. The location servers' private keys are assumed to be distributed to the location servers through a secure channel. The private (secret) key sk is chosen randomly from $\mathbb{Z}_{N/4}$. A public key pk is of the form g^{sk}.
Furthermore, a collision-resistant hash function $H(\cdot)$ is defined which maps strings to $\mathbb{G}$.

2.4.2 Encryption

Encryption of files in LoTAC is carried out by the owner with the help of the localization infrastructure and the cloud storage.

The owner specifies an access set U_F containing the users who are authorized to access the file and a contextual policy P_F with location-time-tag pairs which specify at which time and location the users may decrypt the file. A location is assumed to be the area for which a single location server is responsible. Moreover, he encrypts the file F and the encryption Key K for all users u_j and locations l_i included in the contextual policy:

1. The file is encrypted with a symmetric key K, using a semantically secure encryption scheme:
   $C_F \leftarrow Enc\,(K; F)$
2. Tag-based encryption is used to encrypt the key K for each user separately (user-specific encryption layer). A random $\rho_j \in \mathbb{Z}_{N/4}$ is chosen for each user. An empty tag $\perp$ is used:
   $[\langle \epsilon_j, \kappa_j \rangle, \perp] \leftarrow TB.Enc\left(pk_{u_j}, \perp; K; \rho_j\right) = \left\langle K \cdot H\left(pk_{u_j}^{\rho_j}, \tau\right), g^{\rho_j} \right\rangle$
3. Then, one random $r \in \mathbb{Z}_{N/4}$ is chosen. For each user, the κ_j is encrypted further using ElGamal:
   $\langle \alpha_j, \beta_* \rangle \leftarrow EG.Enc\left(pk_{u_j}; \kappa_j; r\right) = \left\langle \kappa_j \cdot pk_{u_j}^r, g^r \right\rangle$

4. Finally, the location-specific encryption layer is added. The ephemeral key β_*, which is the same for all users, is further encrypted. The encryption is done using tag-based encryption for all the locations which need to be visited. For this purpose, a random $z \in \mathbb{Z}_{N/4}$ is chosen, it is the same for all locations l_i.
   $[\langle \beta, \gamma \rangle, \tau_i] \leftarrow TB.Enc(pk_{l_i}, \tau_i; \beta_*; z) = \left\langle \beta_* \cdot H\left(pk_{l_i}^z, \tau\right), g^z \right\rangle$

5. The owner then uploads to the cloud provider the:
   - The user-specific U_F-ciphertext: $\langle \epsilon_j, \alpha_j, \beta_* \rangle$
   - The location-specific P_F-ciphertext: $\langle \beta, \gamma \rangle$
   - The contextual policy P_F containing pairs of locations and time-tags τ_i at which it is allowed to decrypt the file
   - The encrypted File C_F which can be decrypted with the key K.

2.4.3 User - Location server interaction

The interaction between a user and a location server is an interactive protocol between user u_j, and location server l_i at time t.
u_j has already downloaded the U_F-ciphertext, P_F-ciphertext and C_F as defined in 2.4.2 from the cloud provider. The user is at the location specified by the policy, at the correct time τ_i.
Authentication and localization of the user is assumed to be done correctly by the location server Ls.

1. The user u_j submits to the location server the necessary parts he received from the cloud: $\{\alpha_j, \langle \beta, \gamma \rangle, \tau_i\}$.
2. The location server checks whether the current time t is compatible with the submitted time-tag from the policy, τ_i. If it is not, the location server does not decrypt.
3. The location server then first decrypts the received $\langle \beta, \gamma \rangle$ using the received time-tag as input for the hash-function and its secret key:
   $\beta_* \leftarrow TB.Dec(sk_{l_i}; \langle \beta, \gamma \rangle; \tau_i) = \dfrac{\beta}{H\left(\gamma^{sk_{l_i}}, \tau_i\right)}$
4. To make the user unable to collude with other users, the location server re-randomizes $\langle \alpha_j, \beta \rangle$ with randomly chosen r_{u_j} and the users public key pk_{u_j} thus tieing the decrypted ciphertext to the user:

$\alpha_{ij} \leftarrow pk_{u_j}^{r_{u_j}}$
$\beta_{ij} \leftarrow \frac{\beta_*}{\beta} \cdot g^{r_{u_j}}$

5. Then, the location server sends back the decrypted, re-randomized α_{ij}, β_{ij} to the user.

2.4.4 Decryption

After the user has visited all necessary locations, he is able to decrypt the key K which he may use to decrypt the encrypted file C_F. Decryption is done as follows:

1. First, the user needs to recover κ_j. This is done by multiplying all the received α_{ij} and β_{ij} from the location servers with the α_j and β received from the cloud provider and using the result as an input for ElGamal:
   $\kappa_j \leftarrow EG.Dec\left(sk_{u_j}; \langle \alpha_j \alpha_{ij}, \beta\beta_{ij} \rangle\right) = \frac{\alpha_{ij}\alpha}{(\beta_{ij}\beta)^{sk_{u_j}}}$
2. With the obtained κ_j and the ϵ_j received from the cloud server, the key K may be recovered using tag-based decryption with the empty tag $\perp$:
   $K \leftarrow TB.Dec\left(sk_{u_j}; \langle \epsilon_j, \kappa_j \rangle ; \perp\right) = \frac{\epsilon_j}{H\left(\kappa_j^{sk_{u_j}}, \perp\right)}$
3. At last, the file is decrypted using the key K and the decryption function Dec of a semantically secure encryption scheme:
   $F \leftarrow Dec\,(K; C_F)$

2.5 Security

LoTAC is intended to be secure in the sense that users who are either not part of the access set and/or do not comply with the contextual policy of a file are not able to decrypt that file. It does however not prevent decryption if users share their secret keys. The security argument is based on attacks where users try to decrypt a file either despite not being in the access set or if they are, despite not having visited all necessary locations at the right times. The union of their visited locations however needs to cover the whole policy. The security is proven based on the attack scenario that two users who are both allowed to decrypt the file visit only part of the necessary locations and try to collude to decrypt the file.

Because of the formalism required, this Section is closely based on the respective Sections of [1].

2.5.1 Security Game

Based on the above explained scenario, LoTAC assumes an adversary who chooses the users he corrupts (i.e. the users who collude). The adversary will learn the secret keys of those users.
Furthermore, an oracle mimicing the interaction between location server and user is needed, it is referred to as `ServerDecrypt`. It takes the same input as the location server $(\alpha_j, \beta, \gamma, \tau)$ and in addition the desired location l_i and the public key of the user which interacts with location l_i: pk_{u_j}.
The game itself is defined as follows:

1. Initialization works the same as in LoTAC, the public parameters $g, \mathbb{G}, N, H(\cdot)$ are defined by the verifier and made public.
2. The adversary decides which users he wants to incorporate and gets their private keys handed over.
3. The adversary chooses an access set and contextual policy. For each location in the policy, the adversary further specifies which users do not visit this location at the right time, i.e. do not comply with this pair of the policy.
   The verifier then randomly selects a message m and encrypts it according to the access set and the contextual policy specified by the adversary.
4. The adversary interacts with the `ServerDecrypt`-oracle defined above. His interaction is restricted in the sense that he may not submit a query where the location is part of the contextual policy and the user is in the set of users not complying with the policy for that location.
5. The adversary guesses m' and wins if $m = m'$.

The security of LoTAC is defined in [1] as follows: "LoTAC is secure if there is no p.p.t. algorithm A that has non-negligible advantage in winning the above game".

2.5.2 Reduction Proof

The security of the used tag-based ElGamal encryption scheme is proven by [1] to be TNM-CCA2-secure (as defined in [20]) in the random oracle model under the strong diffie-hellman assumption. Moreover, the security of LoTAC as defined above is proven in the random oracle model under the strong RSA assumption in the group of quadratic residues modulo a safe prime product and the security of ElGamal and Tag-based ElGamal encryption schemes.

Tag-based ElGamal

To prove TNM-CCA2-security of the tag-based ElGamal encryption scheme, [1] construct a simulator S which may break the SDH assumption by interacting with an adversary who has a "non-negligible advantage in breaking TNM-CCA2-security of the scheme". The reduction proof assumes a DH oracle which on input $\langle g^v, g^{uv} \rangle$ decides whether the triple (g^v, g^u, g^{uv}) is a DH-triple. It outputs either 0 or 1.
The simulator S answers two queries: $H(\cdot)$-queries and decryption-queries. It takes as input G, g^u, g^v and outputs g^{uv}.
$H(\cdot)$-queries contain (Y, W) as input. S hands this query over to the DH oracle which either answers with 1 if $(Y, W) = g^y, g^{xy}$ or 0 otherwise. If the answer is 1, S stores g^y, g^{xy} and returns $k = H(g^y, g^{xy})$. If the answer is 0, S returns a random number k.
Decryption-queries contain as input (α, β, τ). S checks if it has stored some $g^y = \alpha$ and if yes, returns α/k. If no, again a random number is returned.

Assuming now S challenges A with the ciphertext $\langle V, R, \tau^* \rangle$, R being randomly selected. Despite being not allowed to query S for a decryption of the challenge, A is assumed to have a non-negligible advantage in breaking the system. Therefore, if the decryption α/k A knows is correct, he must know k. A therefore learned k when issuing a $H(\cdot)$-query for (V, W) for which S did output k because the DH-oracle returned 1. If this is the case, then S stored (V, W) and $W = g^{uv}$ (because the DH-oracle returned 1). Therefore, S can return W to solve the SDH challenge.

LoTAC

We give an overview over how [1] prove the security of LoTAC under the strong RSA assumption in the group of quadratic residues modulo a safe prime product and the security of ElGamal and tag-based ElGamal encryption schemes. They construct a simulator S which breaks the strong RSA assumption under the above mentioned conditions. S uses an adversary A who has a non-negligible advantage in recovering a message of a challenge-ciphertext handed to him by S.

S is defined as follows: It takes as input $N = p \cdot q$ for p, q being safe primes

and $\zeta \in \mathbb{QR}_N^+$ and outputs $\sigma \in \mathbb{QR}_\mathbb{N}^+$ and $e \in \mathbb{Z}_{\mathbb{K}}$ for $\sigma^e = \zeta$.
S answers two queries: ServerDecrypt-queries and H-queries.
ServerDecrypt-queries are defined as in the security game explained in Section 2.5.1. If the received tag is correct, the query responds with $(\alpha_{ij}, \beta_{ij})$ where α_{ij} depends on ζ.
H-queries take as input (χ_1, χ_2, τ) and output either a random h or $\langle e, \sigma \rangle$ that break the strong RSA assumption in $\mathbb{QR}_N^+$. In order for the second output to happen, the input provided must be of the form $\chi_1^{sk_{u_j}} = \chi_2$, $\tau = \perp$ and χ_1 has to fit the U_F-ciphertext that corresponds to the user u_j in a clearly defined way. If this is the case, then (σ, e) can be calculated from the $(\alpha_{ij}, \beta_{ij})$ that S output to A in a legitimate ServerDecrypt query A made before.
Assuming that A has a non-negligible advantage in recovering the message, then he needs to have queried ServerDecrypt with valid arguments for a user in the access set. If this is the case, then S can break the strong RSA assumption.

Chapter 3

Privacy in LoTAC

In this Chapter we will explain how anonymization of users in LoTAC is achieved. We first state the exact goal and assumptions in Section 3.1 and then give a high-level overview over the solution in Section 3.2. After that, an introduction to some prerequisites is given in Section 3.3. In Section 3.4 we explain the first of the two necessary changes to the LoTAC protocol and in Section 3.5 the second change is addressed in detail. At the end, we give a security argument in Section 3.6

3.1 Goal

Our goal is to introduce privacy in LoTAC. This is achieved by making the location server and localization infrastructure oblivious of the user they're talking to. A location server should not be able to differ between two requests from the same user or two requests from different users nor should it be able to know which user made a request.

3.1.1 Attacker Model

We use the same trust assumptions as in the original LoTAC protocol (discussed in Section 2.2.2) with one exception: The location servers are not to identify the users.

We say that we achieved privacy if a location server upon receiving a request from a user is not able to tell who the requesting user is. Moreover, upon a second request, the location server should not be able to decide whether the request was received from the same user or a different one. This should be true as well in the case of collusion, be it among location servers or if a location server colludes with one or more users or even the localization infrastructure.

3.2 Overview

Privacy is an additional feature to the original LoTAC protocol. Thus, we want the changes to the original protocol to be minimal. However, since the original LoTAC protocol relies on the GSM network to achieve user authentication and the authentication of SIM cards can not be turned off in these networks, we are not able to use GSM networks when we want the user to stay anonymous. We are however free to use any other networks where the user is not identified such as e.g. certain wireless LANs or bluetooth. This follows from the fact that with GSM, a user's identity is tied to the hardware he uses to connect to the network (i.e. the SIM card). Moreover, with GSM, the hardware identifier ("IMSI") may not be easily spoofed. Therefore, the IMSI ties the user to his identity, i.e., phone number. In wireless LANs or bluetooth networks however, the account a user uses for e.g. connection authorization is most of the time not tied to the hardware; he may use the same account on different computers. As for spoofing, the hardware identifier ("MAC Address") of a network or bluetooth card may be quite easily spoofed. Therefore, it is reasonable to assume that user privacy may be achieved in some wireless and bluetooth networks.

In order for the user to stay anonymous, only his interaction with the location server needs to be changed. In the original LoTAC protocol, because of the authentication necessary in GSM, the user is known by the location server which enables the location server to get the users public key from the localization infrastructure. Afterwards, the location server uses the users public key to re-randomize the decrypted ciphertext. This is required to prevent colluding users.
In our setting however the user is not known to the location server, therefore the user needs to send his own public key to the location server without being authenticated in any way. This raises two problems. The first one is that we do not want to trust the user to send the correct public key to the location server. The second problem is that we do not want the location server to be able to assign two different requests to the same user solely by comparing the submitted public keys.
The first problem is solved by introducing a zero-knowledge proof of knowledge. This enables the user to prove that he knows the secret key which belongs to the presented (randomized) public key. The proof of knowledge we use was introduced by Okamoto [22]. The details are described in Sec-

tion 3.5.
The second problem is solved by randomizing the public key of the user before submitting it to the Location Server. We explain the solution in detail in Section 3.4.
The rest of the protocol is the same as in the original LoTAC protocol.

3.3 Prerequisites

In this Chapter, we assume knowledge of zero-knowledge proofs which we will briefly introduce here.

3.3.1 Zero-Knowledge Proofs of Knowledge

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff [14]. They are used to prove knowledge of a secret without handing over the secret, i.e. after the proof, the verifier believes that the prover knows a secret variable x, but the verifier does not learn x. The proof consists of an interactive protocol between the prover and the verifier where the verifier could guess the answer with (in our case) 50% probability of correctness if he were cheating. Thus, the protocol is repeated multiple times until the chance of success for the prover is minimal in case of cheating.

An interactive proof is said to be a proof of knowledge if it satisfies two properties: completeness and soundness. Completeness denotes the fact that every true statement has a proof. The property of soundness is met if only true statements have a proof. Proofs of knowledge are said to be zero-knowledge if the verifier, while carrying out the proof, does not learn any information he did not already know before the proof. More details may be found for example in the lecture notes for the course "Cryptographic Protocols" by Prof. Ueli Maurer [15].

Example 3.3.1. The protocol we use in Chapter 3 was introduced by Okamoto [22]. Here, as an example of how zero-knowledge proofs look like, we introduce Schnorr's protocol:

The goal is for the prover to prove that he knows a secret variable x. Publicly known variables are g which is a generator of the cyclic group $\mathbb{G}$ and $z = g^x$.

1. The prover chooses a random variable $r \in \mathbb{G}$ and sends to the verifier $t = g^r$.
2. The verifier chooses a challenge $c \in \{0, 1\}$ at random and sends it to the prover.

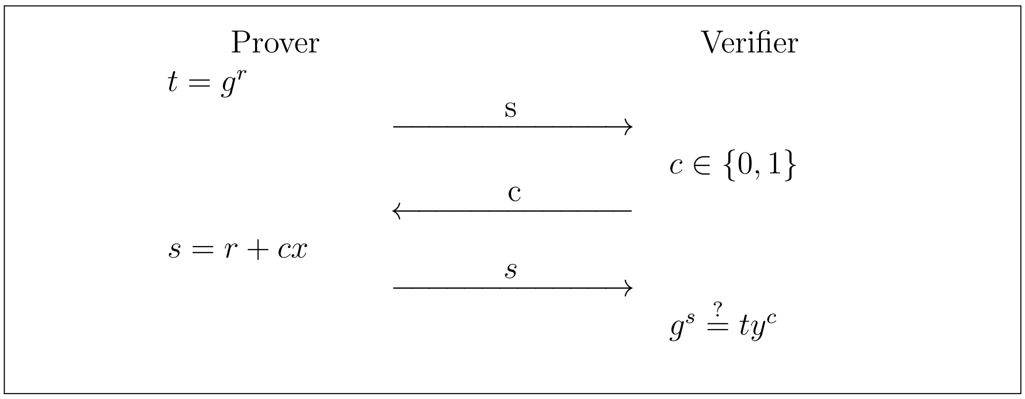


Figure 3.1: Schnorr's Protocol

3. The prover calculates $s = r + cx$ and sends it over to the verifier.
4. The verifier checks whether $g^s \overset{?}{=} ty^c$.

If the equality is correct, then the prover has successfully shown knowledge of x.

We use a notation which makes use of arrows to symbolize messages to be exchanged between the prover and verifier. The protocol explained above would then look as in figure 3.1.

3.4 Blinding of a Users' Public Key

In this Section, we address the problem that the location server needs a user's public key to randomize the decrypted ciphertext, but that we do not want to hand over the user's public key because that would identify the user.

Overview To overcome the problem that a user may be identified based on his public key, the user randomizes his public key before handing it to the server. Upon receiving the decrypted, re-randomized ciphertext from the location server, a user is then able to de-randomize his public key.

To avoid confusion with the re-randomization of the ciphertext which is done by the location server, we define the randomization a user applies to his public key as "blinding". The idea was originally introduced by Chaum [12]. To blind his public key pk the user first chooses a random

$r'' \in \mathbb{Z}_{N/4}$. The actual blinding is then done by calculating $pk \cdot h^{r''}$, where h is a generator of $\mathbb{G}$.

3.4.1 Changes to LoTAC

During the setup phase, the localization infrastructure needs to chose a second generator of $\mathbb{G}$ which we call h.
When the user arrives at the desired location, before starting interaction with the location server, he blinds his public key using the second generator and a randomly chosen variable. He submits the ciphertext and his blinded public key to the location server.
The location server then decrypts the provided ciphertext with its private key and re-randomizes the recovered ciphertext with the blinded public key of the user. He returns the re-randomized ciphertext to the user.
The user then un-blinds this re-randomized ciphertext before decrypting it with his private key.

In order to execute these steps as described, some changes to the original protocol of LoTAC are necessary. Most of the changes happen in the phase of the user interacting with the Location Server. An overview over the whole protocol is given in figure 3.2.

Localization Infrastructure

When setting up the encryption, the Localization Infrastructure needs to choose a second generator h of $\mathbb{G}$ in addition to g which is later used by the user to blind his public key. This second generator is publicly known.

The protocol for encrypting a file is not changed. Furthermore, the process for a user to get the file from the cloud is the same. The new interaction between user and Location server takes place as explained below.

Blinding the Public Key

After downloading the ciphertext from the cloud, the user travels to the desired location. But before interacting with the location server there, he blinds his public key as defined in Section 3.4. The user does then not only send the ciphertext and time-tag to the location server, but his blinded public key as well.

Location Server

The location server acts very similarly as in the original LoTAC protocol. First of all, it does not need to identify the user to get his public key. Instead, the blinded one which it received from the user is used. Furthermore, the user does not submit α_j because it is not used (only the public key is re-randomized) and since it is specific for a certain user, it would break anonymity. No changes are necessary to the way the location server handles decryption. At the end, the location server returns the randomized α''_{ij}, β_{ij} as in the original protocol and additionally returns $h^{r'}$.

Un-Blinding the Public Key

Before the user can decrypt the key, he needs to un-blind his public key. Un-blinding is done by dividing the randomized α_{ij} by the randomized and blinded h: $\frac{\alpha''_{ij}}{h^{r' \cdot r''}} = \frac{pk^{r'} \cdot h^{r'' \cdot r'}}{h^{r' \cdot r''}} = pk^{r'} = \alpha_{ij}$. This randomized, but un-blinded α_{ij} can now be used to recover the key K.

Recovery of the key

The user is now ready to decrypt the symmetric key which the owner used to encrypt the file. This is done the exact same way than in the original LoTAC protocol, thus exactly as explained in Section 2.4.4.

3.5 Proof of Knowledge of the Secret Key

To prove to the location server that the submitted pk is indeed the user's pk, we use a proof of knowledge of a representation proposed by okamoto [22]. This proof of knowledge enables the user to prove that he knows the secret key which belongs to the blinded public key he presented to the location server.

The protocol takes place between the user and the location server. The publicly known parameters are g and h which are generators of $\mathbb{G}$, the blinded public key $pk'' = g^{sk}h^{r''}$ and ν which is defined as $\nu = g^{-sk}h^{-r''}$. The users secrets are sk and r'' which form the blinded public key.
At the beginning of the protocol, the prover (user) chooses two random variables r_1 and r_2 and calculates $s = g^{r_1}h^{r_2}$ which is sent to the verifier (location server). The location server then chooses a random variable $e \in \{0, 1\}$ which it sends back to the user. The user on his turn calculates and sends the location server two variables y_1 and y_2 containing the two secrets.

User | Ls

Downloaded from CP:

$U_F : \langle \epsilon_j, \alpha_j, \beta_* \rangle$.

$P_F : \langle \beta, \gamma \rangle$

C_F: Enc(K;F) [1]

Blinding of pk_u:

$r'' \in \mathbb{Z}_{N/4}$

$pk'' = pk \cdot h^{r''}$

$$\xrightarrow{\{\langle \beta, \gamma \rangle, \tau_i, pk''\}}$$

$\boxed{\text{The user proves to Ls that the submitted, blinded } pk'' \text{ indeed belongs to him using the proof of knowledge explained in Figure 3.3}}$

$\beta_* \leftarrow TB.dec(sk_l; \langle \beta, \gamma \rangle, \tau_i)$

Re-randomization of pk'' and β_*:

$\alpha''_{ij} \leftarrow (pk'')^{r'}$

$\beta_{ij} \leftarrow \frac{\beta_*}{\beta} \cdot g^{r'}$

$$\xleftarrow{\alpha''_{ij}, \beta_{ij}, h^{r'}}$$

Un-blinding of pk_u:

$\alpha_{ij} \leftarrow \frac{\alpha''_{ij}}{h^{r' \cdot r''}}$

Recovery of the key K:

$\kappa_j \leftarrow \frac{\alpha'}{\beta'^{sk}}$

$K \leftarrow TB.Dec\left(sk_{u_j}; \langle \epsilon, \kappa_j \rangle; \perp\right) = \frac{\epsilon}{H\left(\kappa_j^{sk_{u_j}}, \perp\right)}$

Figure 3.2: LoTAC protocol with changes

$$\begin{array}{lcl}
\text{User} & & \text{Ls} \\
s = g^{r_1} h^{r_2} & & \\
 & \xrightarrow{\text{s}} & \\
 & & e \in_R \mathbb{Z}_2 \\
 & \xleftarrow{\text{e}} & \\
y_1 = r_1 + e \cdot sk & & \\
y_2 = r_2 + e \cdot r'' & & \\
 & \xrightarrow{y_1, y_2} & \\
 & & s \stackrel{?}{=} g^{y_1} h^{y_2} v^e
\end{array}$$

Figure 3.3: Proof of Knowledge of the Secret Key

At the end, the location server compares the received values to the initially sent variable s: $s \stackrel{?}{=} g^{r_1+e\cdot sk} h^{r_2+e\cdot r''} g^{-sk\cdot e} h^{-r''\cdot e}$.
The probability of a cheating user to win is $\frac{1}{2}$. It has to be repeated a sufficient amount of times in order for the probability of a cheating user to be negligible.

3.6 Security

There are two variables we do not want the location server to know, but need to show him somehow during the above presented protocol. First, the public key of a user and second, the secret key. In this Section we argue why the location server does not get to know the identity of the user despite the user showing the location server his blinded public key and proving knowledge of the secret key.

The protocol uses several random variables. We would actually want to choose random values from $\mathbb{Z}_{\phi(N)}$. But to be capable of doing so, we would need to know the group order $\phi(N)$ of $\mathbb{G} = \mathbb{QR}_N$ which is $\phi(N) = (p-1)(q-1)$ for $N = pq = (2p'+1)(2q'+1)$. Since the users do not know the factorization of N, it is not possible for them to calculate $\phi(N)$ and choose random elements from $\mathbb{Z}_{\phi(N)}$. As a substitute, we use $\mathbb{Z}_{N/4}$. This is a close enough estimation, since $(2p'+1-1)(2q'+1-1) = 4p' \cdot q'$ (explained for example by Lucks [18]).

As explained in Section 3.4, we blind the public key of a user by calculating $pk'' = pk \cdot h^{r''}$ where r'' is chosen uniformly at random $\in \mathbb{Z}_{N/4}$. Because of that, $h^{r''}$ is random in $\mathbb{G}$ and therefore pk'' is random in $\mathbb{G}$ as well, and thus blind for the location server.

Knowledge of the secret key is proven by the proof of knowledge explained in Figure 3.3. The reason for the proof not leaking any information about either the secret key sk nor the randomness r'' follows directly from the zero-knowledge property of the proof as explained in Section 3.3.1 and which was proven for our protocol by okamoto [22].

As argued above, LoTAC with our extension provides anonymity of the user. But apart from the user identity, how much of the information are we able to keep secret from the location servers? We do not only submit the public key to the location server, but β, γ and the time-tag τ_i as well. β and γ are file-specific, therefore by sending them to the location server, it can at least match two requests for the same file, even if it does not know whether they originate from the same user or not.
Moreover, assuming that quite a big part of requests are valid and the location server is colluding with other location servers, then it is able to reconstruct the policy of a file. No location server will be able to learn whether a single tag is valid, but if a single tag is requested a lot of times for certain β and γ, then it is reasonable to assume that the tag is valid.
So, even if the a single user is anonymous to the location server, it is worthwhile to remember that a file is not.

Chapter 4

Implementation

The second part of this thesis consists of implementing the LoTAC protocol as an iPad application in Objective-C. Furthermore, the already existing server has been modified and a second server acting as cloud server was implemented in Python.

In this chapter the implementation of the LoTAC system is introduced. It is divided into three Sections which explain the Setup (Section 4.1), the two servers (Section 4.2) and the client (Section 4.3). Each of the Sections first describe what the respective part of the system does and then, in their "Execution" Subsections, how to run it.

4.1 Setup

4.1.1 Implementation

The python script *setup.py* represents the Localization Infrastructure which sets up the encryption parameters and public/private keys of users and Location Servers.
The script has a number of safe primes hardcoded to choose from. Safe primes must be generated using openSSL. The most convenient way is to use the correspondent function in the iPad Client (see Section 4.3.2).

It first calculates N using two safe primes. Then, it looks for a generator of the group QR_N^+ which is referred to as g. g and N are called the public information.

The next task includes the generation of public and private key pairs of users and location servers. Private keys (i.e., secret keys, sk) are chosen at random and public keys are calculated by raising g^{sk}. A username respectively location-name is given and the generated information saved.

At the end of the script, the saved information is encoded into json and five files are written:

`locInfo.txt:` an input file for the server, containing a dictionary with the location's public and private keys. The entries are of the form
$\{loc1 : \{sk : 1234, pk : 1234\}, loc2 : \{sk : 1234, pk : 1234\}, ...\}$.

`userInfo.txt:` used by the server,containing a dictionary with the users' public keys. The entries are of the form
$\{user1 : 1234, user2 : 1234, ...\}$.

`pubInfo.txt:` loaded by the server and contains the values of g and N in the form
$\{N : 1234, g : 1234\}$.

`clientInput.txt:` an input file for the client, containing an array with the users' public and private keys. The entries are of the form
$[\{uID : user1, uK : [1234, 1234]\}, \{uID : user2, uk : [1234, 1234]\}, ...]$.

`loclist.txt:` loaded by the client, containing an array of available locations as strings. It is of the form:
$[loc1, loc2, ...]$.

4.1.2 Execution

Because finding a generator includes a for loop over a very large integer, the script must be run using python3. It can be started from a terminal using the command '`python3 ./setup.py`' with the working directory being the folder '`lotac_server/setup/`'.

The files which are being generated must be handled as follows:

- **locInfo.txt, userInfo.txt** and **pubInfo.txt** are to be loaded by the server. They are automatically copied to the folder `lotac_server/` (i.e., one level higher in the folder hierarchy) where they are expected to reside by the server. Nothing needs to be done with them.
- **clientInput.txt** and **locList.txt** are to be read by the client. They must be moved manually to the `Resources` folder in the LoTACclient Xcode-project.

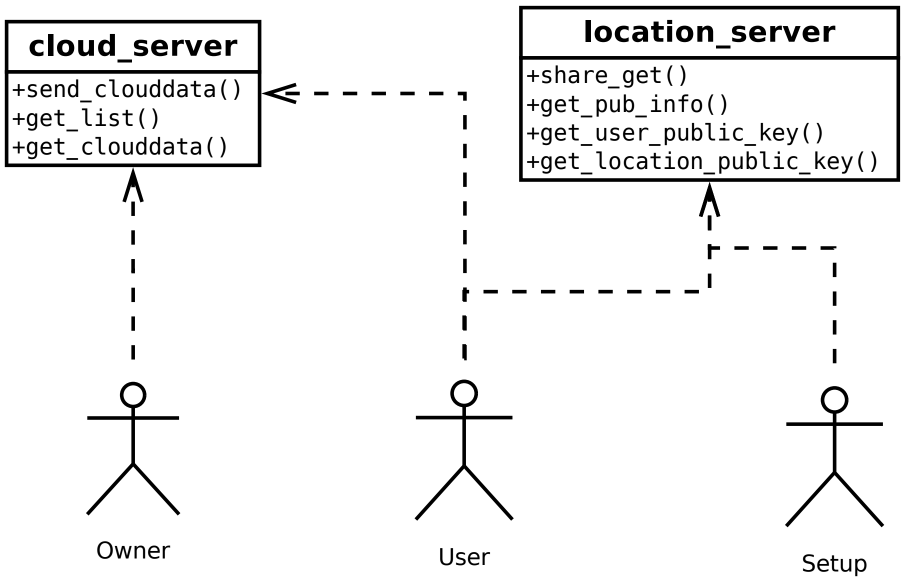


Figure 4.1: Functionality of the two servers

4.2 Server

The system contains two servers, one acting as the location server and localization infrastructure and a second server acting as the cloud server. Both are written in python. An overview over the functionality that the two servers provide is shown in Figure 4.1.

4.2.1 Location Server

The location server is based on previous work where the LoTAC protocol was implemented as an Android application. The server has been completely revised in order to comply with the protocol as specified in the LoTAC paper [1].

Design

The server consists of four files: The XML-RPC library, a wrapper class `server.py` which starts the server and `MLTR.py`, which provides the actual functionality and a class `Utils.py` containing utility functions needed by `MLTR.py`.

The server and client communicate with each other by means of XML-RPC over HTTPS. The library used was included in the server code used to build LoTAC for Android, it was not changed for this Thesis. It is run using python2.

Implementation

Wrapper class: `server.py`. The wrapper class specifies the address and port the server listens on and specifies the class where the actual functionality is implemented (`MLTR.py` in our case). Furthermore, it tells the server which certificate files to use. Finally, it starts the server.

Server functionality: `MLTR.py`. The services the server provides and its initialization are implemented in the `MLTR.py` file. It contains the following functions:

`__init__` is the function which initializes the server. It declares three dictionaries, one for the locations' public and private keys, one for the users' public keys and one for the public Information, e.g. N and g. It then fills these dictionaries by reading the relevant information from the files which were generated by `setup.py`: `pubInfo.txt`, `locInfo.txt`, `userInfo.txt`.

`share_get` is the function which acts as location server. It takes as arguments a location-name, a username, a time tag and the encrypted γ. Its task is to remove one layer of encryption, namely the one which ensures that the user really did visit a certain location at the correct time. To achieve this, it does a tag-based decryption of γ using the secret key of the given location and re-randomizes the decrypted ciphertext using the public key of the given user. It returns α_u (the re-randomized public key) and β_u (the re-randomized, decrypted ciphertext) as explained in Section 2.4.3.

`get_pub_info` simply returns N and g to the caller. It does not take any arguments. It is only used during the setup of the client.

`get_user_public_key` takes a list of users and returns a dictionary of these users' public keys. It is used by the owner (i.e. in the Client) during encryption.

`get_location_public_key` serves the same purpose for locations. It takes a list of locations and returns a dictionary containing these locations' public keys. It is used by the owner (i.e. in the Client) during encryption as well.

Utility class: `Utils.py` There are some small functions which are needed for the server to serve requests. This class was provided with the old server and we did only minor changes to it and removed some unused functions.

It contains a function returning a random number of a certain bit length and below a certain limit. The function just calls the usual `random.randint()` with appropriate arguments.
The last two functions are used to calculate the modular inverse of a number: euclidean gcd to calculate it and a wrapper function. The wrapper function calls the `gcd` function with appropriate arguments and checks for the result's validity before returning it.

4.2.2 Cloud Server

The cloud server takes the role of the cloud storage in the LoTAC protocol. It receives the U_F-ciphertext, the P_F-ciphertext and the contextual policy P_F from the owner. Since the client does only encrypt the key without encrypting a whole file, the cloud storage does not need to receive an encrypted file from the client.
The cloud server serves requests from users who want to decrypt files: It hands out a list of files intended for that user and upon selection by the user, makes available the requested U_F-ciphertext, P_F-ciphertext and P_F-policy.

Design

Similar to the location server, the cloud server relies on the XML-RPC library which was part of the server of the Android app. Moreover, there is the wrapper class `cloud_server.py` which instantiates and starts the server and the protocol class `Cloud.py` which provides the functionality described below.

Implementation

The following functions are provided by the cloud server. No authentication is done upon request; anyone may request or send data.

`send_clouddata` receives the encrypted data from the owner. It takes as arguments a list of usernames (i.e. the users in the access set U_F), a dictionary with key `username` and value the U_F-ciphertext for the respective user, a list P_F-policy with location/time-tag-pair entries and P_F-cipher, the location-specific ciphertext.

`get_list` returns a list of files which may be decrypted by a given user. It takes as argument a username and returns a list of strings.

`get_clouddata` returns the encrypted data for a user so he can decrypt it. It takes as argument a username and a filename (a list of valid filenames may be obtained by using the `get_list` function). It returns the P_F-ciphertext, P_F-policy and U_F-ciphertext.

4.2.3 Execution

Both servers are run from the terminal by issuing the command '`python2 ./server.py`' and '`python2 ./cloud_server.py`', assumed that the working directory is the folder '`lotac_server/`'. The servers start on localhost, listening on ports 4444 (cloud server) and 8888 (location server).

In order to be started successfully, the location server requires the files `locInfo.txt`, `userInfo.txt` and `pubInfo.txt` to be in the same working directory. For more information on these files, see Section 4.1.

4.3 Client

The client side of the implementation is built as an iPad application using Objective-C and Xcode version 4.6.2. The app provides the owner and the user functionality as well as a small tool to generate safe primes. Generation of safe primes is only rarely used for the setup of the server.

4.3.1 Design

The design of the client is based on the Model-View-Controller Pattern. There are three different views which are explained in detail below. Each of the views has an associated controller-class. The model-part which implements the protocol consists of the three core-protocol classes `SetupProtocol`, `OwnerProtocol` and `UserProtocol` representing the setup-, owner- and user-part of the protocol and of the helper classes `Pair`, `ServerQuery`, `PublicInfo` and `ServerOwner`. The latter is the superclass of the core-protocol classes, providing the interface to connect to the server in order to place requests and receive replies. `ServerQuery` represents a request to the server. `PublicInfo` stores publicly available parameters of the ElGamal encryption scheme which are provided by the setup. `Pair` provides the functionality of a pair of two big numbers. The design of the model-part of the app is summarized in Figure 4.2, some operations are simplified, others left out.

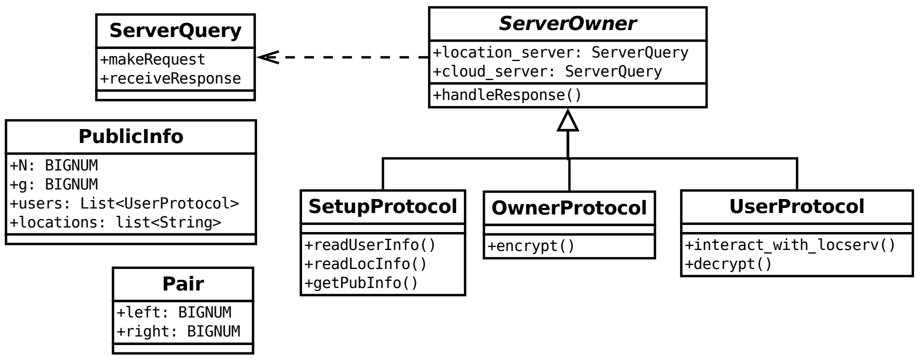


Figure 4.2: Design of the Client

4.3.2 Implementation

Libraries

Two external libraries were used in the development of the client: First openSSL which provides big numbers and the possibility of modulo operations for an integer. Second, to communicate with the XML-RPC server, an XML-RPC-client library is used.

openSSL is an open-source cryptography and SSL/TLS toolkit [23]. It is licensed under an apache-style license. In this project, we use the newest version available at the time of implementation, 1.0.1e, dating from february 11th, 2013.

Moreover, we use a project skeleton from Michael Tyson and Stephen Lombardo [27] called openSSL-xcode which enabled us to compile the openSSL library to use as a static library within our Xcode-project.

The XML-RPC client for iOS by Dallas Brown [9] is a fork of the XML-RPC project by Eric Czarny [1]. It is licensed under a MIT-license.

It provides the functionality necessary to issue requests to and handle responses from an XML-RPC server using HTTPS.

Setup of iPad-App

By pressing the button “Setup” as seen in Figure 4.3a, the client loads all necessary information to act as owner and client. First, it reads from a file all the users’ private and public keys and uses them for instantiation of

[1]to be found at https://github.com/eczarny/xmlrpc

the available users. For practical reasons, the available users are saved in the class `PublicInfo` so it is possible to easily choose a user for decryption. Furthermore, the application reads a list of available locations from a file. These files are generated by the python setup routine described in Section 4.1 and have to be put in a folder named “Resources” in the root of the client’s Xcode-project. Then, the server is contacted and asked for the publicly known variables g and N which are then saved for the `OwnerProtocol` and `UserProtocol` to use by the class `PublicInfo` .

Owner

To act as an owner, a user has to take three actions. First, he has to choose a set of users for which to encrypt a file for, then he has to choose a policy (i.e. a list of locations and time frames at which the file may be decrypted) and at last, he has to press the button named “Start encryption”.

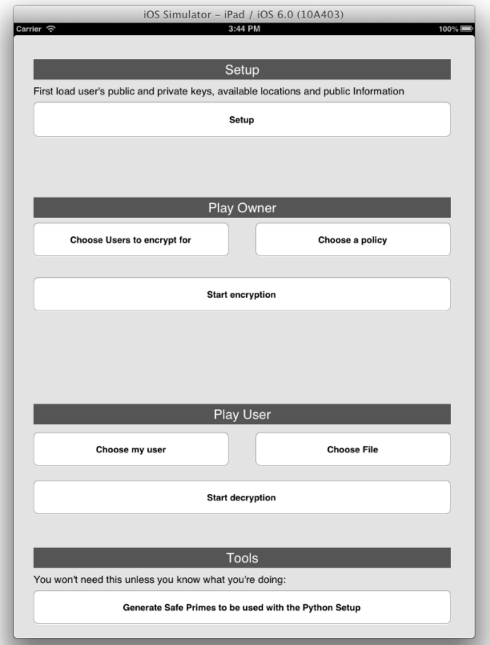
Choosing a set of Users Upon pressing the button named “Choose users to encrypt for”, a view with a list of available users opens (see Figure 4.3b). By tapping on a user, a checkmark appears on the right (a second tap disables that checkmark). Several users may be chosen that way. If all the desired users carry a checkmark, the “OK”-button can be pressed.

Choosing a policy Upon pressing the button named “Choose a policy”, the view seen in Figure 4.4a opens. At the beginning, it contains an empty list, three so-called pickerwheels, a button with a plus and one with a minus sign, and an “OK”-button. A location/time tag pair is selected by rotating the pickerwheel at the left to the desired location, then in the pickerwheel in the middle selecting the start time of the desired time frame in which it is allowed to decrypt the file, and at last in the pickerwheel on the right, the duration of the time frame in hours has to be selected. By pressing the plus-button, the selected values are added to the list as a “location/time tag”-pair in the policy.

Editing of policies is not possible, but wrongly entered location/time tag pairs may be deleted by selecting them in the list (a checkmark appers on the right) and then tapping the minus button. Tapping the minus button deletes all currently selected list entries.

Upon pressing the “OK”-button, the currently visible location/time-tag pair-list is returned as the selected policy.

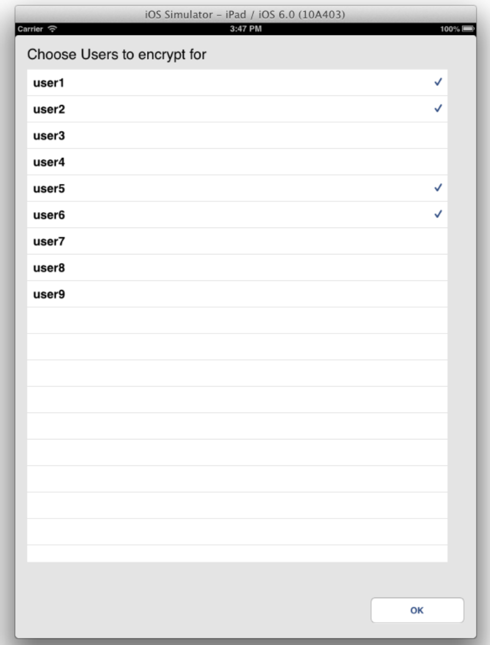
The views to choose users and a policy always start up empty, they do not show what has been selected the last time they were used.



First load user's public and private keys, available locations and public Information

You won't need this unless you know what you're doing:

(a) The main view



(b) The "Choose Users" view

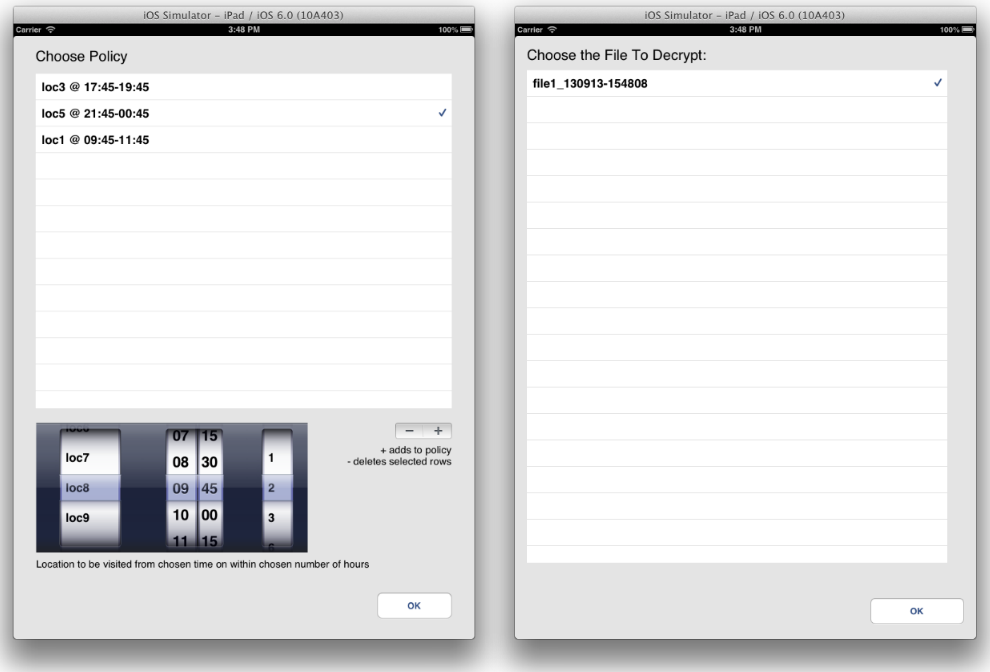
Figure 4.3: Views of the iPad application

The button "Start encryption" instantiates a new owner and hands him over the list of chosen users and the policy. The owner then encrypts a key and uploads the ciphertext along with the policy to the cloud server. Moreover, once encryption is finished, the owner is discarded and next time "Start encryption" is pressed, a new one is instantiated. This is fine since we do not need to know the owner of a file in order for the rest of the protocol to succeed.

User

The user functionality requires three actions to take as well. First, we choose a user. Second, we select a file to be decrypted and at last, the file (i.e. the key to decrypt this file) is decrypted by pressing the button "Start decryption".

User selection By pressing the button "Choose my user" we decide which user to play the protocol with. Upon pressing, a new view opens, containing


iOS Simulator – iPad / iOS 6.0 (10A403)
Choose Policy
loc3 @ 17:45-19:45
loc5 @ 21:45-00:45 ✓
loc1 @ 09:45-11:45
loc7
loc8
loc9
+ adds to policy
- deletes selected rows
Location to be visited from chosen time on within chosen number of hours
OK
iOS Simulator – iPad / iOS 6.0 (10A403)
Choose the File To Decrypt:
file1_130913-154808 ✓
OK

(a) The “Choose Policy” view (b) The “Choose File” view

Figure 4.4: Views of the iPad application (cont.)

a list of available users (as read in by the setup). By tapping on a user, this user is selected (a checkmark on the right appears). Only one user may be selected. Selecting a second user automatically un-selects the first one. If the selection is satisfying, we confirm the selection the “OK”-button and return to the main screen.

File selection Then, we select a file to decrypt. We do this by pressing the button named “Choose File”. This starts a query to the server for a list of files which may be decrypted by the chosen user. Upon receiving an answer, a view opens which shows the received list. If no user is selected, no view opens and an error is shown on the logging console (but not the UI).
Again, as with user selection, one file may be selected. Upon selection of a second file, the first checkmark disappears. If the selection is made, we press the “OK”-button to confirm and return to the main screen.

Decryption At last, after a user and a file have been selected, we are ready to decrypt. Pressing the button named “Start decryption” hands over the selected file to the chosen user and lets him start decryption. Possible output is seen in the logging console. The correct decryption is ensured by using an assert-statement which checks correctness of the decrypted key.

Tools

Generating Safe Primes For the setup of ElGamal encryption, we need to generate two safe primes p' and q' (remember, the modulus N equals $(2p'+1)(2q'+1)$). Since openSSL provides a function to generate safe primes, for convenience reasons, we introduced a button into the application which calls this function. The python setup script already provides a few hardcoded safe primes to select from, so it is not absolutely needed to use this button before setting up the server. However, in case a new N is desired, this button may be used to generate the underlying safe primes.

Upon pressing the button, no view is opened. The output is given on the logging console, the safe primes must then be copied into the python setup script manually. If necessary, the number of bits of the safe prime may be changed in the code. The location to change this is to be found in the file `ViewController.m`, in the method `genSafePrimes:sender`, at the row `int nrBits = 128` (the number — in this case 128 — has to be changed to the desired length). If N (and thus the public and private keys as well) should have a length of 2^m bits, then the number inserted should equal to 2^{m-1}.

4.3.3 Execution

The iPad application (i.e. “the client”) may either be run in the simulator, or on an iPad. In the second case, the code needs to be signed which requires a developer account and an iPad associated with it. The app does not run on other iPads because it is not in the App Store.

To start the iPad app, it is necessary to setup the project in Xcode.
To include the libraries which are mentioned above, the following linker tags have to be added: `-lcrypto -lssl -lXMLRPC_iOS`. To ensure the connection between the app and the server, the IP of the server has to be manually entered into the code, namely in the respective constructors of the `SetupProtocol`, the `OwnerProtocol` and the `UserProtocol`.
Furthermore, as explained before in Section 4.1, the necessary files need to be placed in the “Resources”-folder of the project. It has to be made sure

that they are mentioned under “Copy Bundle Resources” in the tab “Build Phases”.

After the setup is complete, in the “Scheme” dropdown at the top left of the window, one can choose whether to run the app within a simulator or on the iPad. Pressing the “Run”-button starts the app.

Chapter 5

Measurements

We explain how we tested our implementation in Section 5.1 and present the results in Section 5.2. In Section 5.3 we then analyze and interpret the results.

5.1 Setup

The experiments are conducted on an iPad mini and a MacBook Pro. The iPad comes with a 1GHz dual-core ARM processor and 512MB RAM running iOS 6.1. We used the Xcode tool “Instruments” which is able to automate UI interaction using java-script. The MacBook Pro ran the server which was connected to the iPad. It runs on a 2.4GHz Intel Core 2 Duo processor and has 4GB of RAM. The Mac OS X version is 10.7.5. The iPad connected to the server through WLAN.

The java-script which was used to automate the UI interaction first taps on the setup button once and then conducts the owner-part and user-part of the protocol for different numbers of users and locations, i.e. chooses a different number of users to encrypt for and a different number of policy entries. The user for which we decrypt is always only one user. The number of users and locations used were 1, 5, 10 and 20, thus 16 experiments per run. Every run was conducted three times per key size. Key sizes required new input files to be generated, therefore manual handling was necessary. 4 key sizes were tested: 128, 256, 512 and 1024 bit respectively. Therefore we conducted a total of $4^3 = 64$ different experiments three times.

Execution times were measured by logged timestamps. The logs were then

Encryption	128 bit	256 bit	512 bit	1024 bit
User portion	3.34 ms	10.99 ms	53.99 ms	317.8 ms
Location portion	1.69 ms	5.37 ms	26.68 ms	157.85 ms
Total	5.03 ms	16.36 ms	80.68 ms	475.65 ms
Decryption				
User portion	3.30 ms	7.61 ms	32.61 ms	180.32 ms
Location user side	0.05 ms	0.06 ms	0.09 ms	0.16 ms
Location server	0.51 ms	1.29 ms	4.84 ms	26.35 ms
Total	3.35 ms	7.67 ms	32.69 ms	180.48 ms

Table 5.1: Duration of encryption and decryption for different key sizes, one user and one location

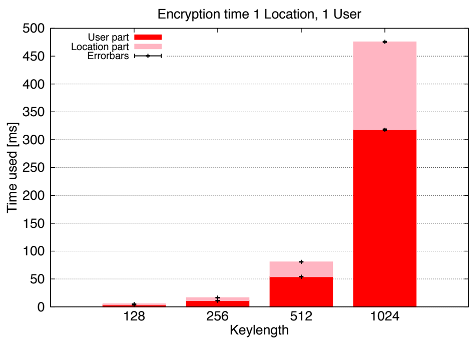
parsed by a python script which wrote the necessary values into a data file used as input for `gnuplot`. Every experiment contained four measured times: The encryption time used to encrypt for users, the encryption time used to encrypt for locations, the decryption time used to decrypt for locations and the decryption time used for the user (which was always one despite the number of users which the file was encrypted for). On the server, we measured the time needed to handle a share, i.e., remove the location layer of the ciphertext. Since the server does not distinguish between a ciphertext which was only encrypted for one location and one that was encrypted for multiple locations, we multiplied the mean time needed to handle a share by the number of locations in Figure 5.2 and Table 5.2.

5.2 Results

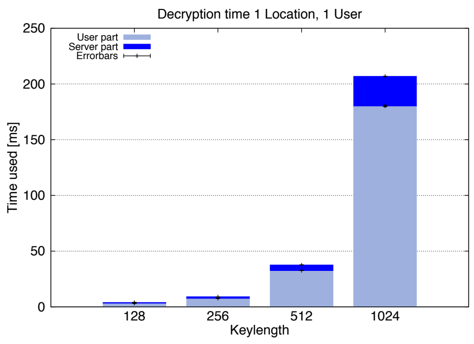
We will first have a look into the differences in the times used to encrypt and decrypt, depending on the key size and number of locations. Section 5.2.2 shows detailed results of encryption times with varying number of users and locations.

5.2.1 Encryption and Decryption

As Figure 5.1 compares the time used for encryption and decryption for different key sizes and one user/location respectively. The plot shows, that across all key sizes, encryption for one location and one user takes generally longer than decryption for one location. The measured times range from 5ms for 128 bit keys to 475ms for 1024bit keys, both when encrypted for one location and one user. Decryption for one location and one user on the



(a) Encryption



(b) Decryption

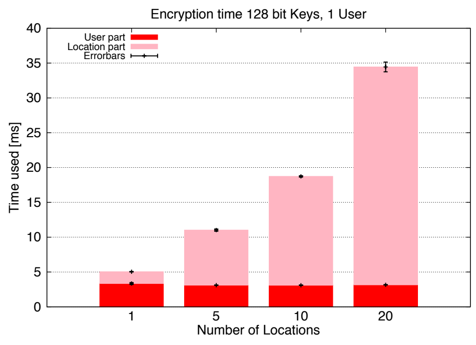
Figure 5.1: Duration of encryption and decryption for different key sizes

Decryption	1 Location	5 Loc.	10 Loc.	20 Loc.
User portion	3.30 ms	3.56 ms	3.21 ms	3.03 ms
Location user side	0.05 ms	0.46 ms	0.99 ms	2.36 ms
Location server	0.51 ms	2.04 ms	5.08 ms	10.16 ms
Total	3.86 ms	6.04 ms	9.28 ms	15.55 ms

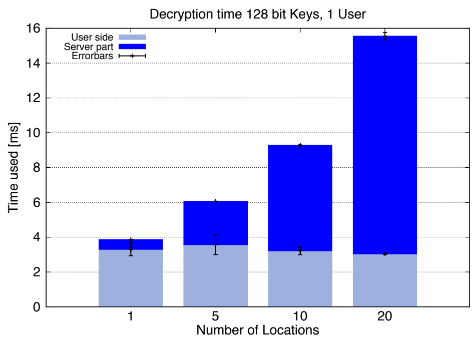
Table 5.2: Duration of decryption for different number of locations, one user and 128 bit keys

other hand ranges from 3.35ms for 128 bit keys to 180ms for 1024bit keys. These results can be seen in Table 5.1˙
When encrypting, the owner (i.e., iPad application) is taking care of both the location part and the user part of the encryption. When decrypting, the location part is shared between the location server which removes the location layer of the encryption and the user which is handling the received shares. The server is only able to measure the time it needs to handle a single share, because it does not keep track of multiple location server requests for a single file. Therefore, in Table 5.3 we just multiply the time for handling a single share by the number of locations needed. The measurement of the location server for a single location is the mean over all measured occurences of the server handling a single share, which are several hundred. The user part of the decryption is handled entirely by the iPad application. We did not take into account the network delay between the server and the client, which would of course change this ratio considerably in everyday use.

Figure 5.2 compares the time used for encryption and decryption for different number of locations and key size 128 bit. We did measure encryption and decryption times for 5, 10 and 20 users and locations respectively. As



(a) Encryption



(b) Decryption

Figure 5.2: Duration of encryption and decryption for different number of locations

said before, decryption is by design always done for one user, so we do not compare the results for the encryption time of more than one user to the decryption time.
As we can see in the plot, the time needed to encrypt for one user and one location is 5 ms. Decryption of this ciphertext takes a bit less than 4 ms, which is about 20% less. This difference gets larger when we encrypt for more locations: to encrypt for one user and 20 locations with keys sized 128 bit takes 34.45 ms whereas decryption only takes 15.55 ms which is about half the time. We give some reasons for this behavior in Section 5.3.

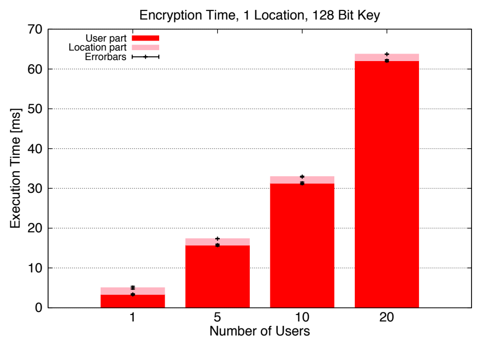
5.2.2 Number of Users and Number of Locations

Figures 5.3 and 5.4 show the time needed for encryption when 128 bit keys are used. These plots are representative for other key sizes as well since the linear increase in time when the number of users/locations is varied stay the same for other key sizes. The mentioned results are summarized in Table 5.3

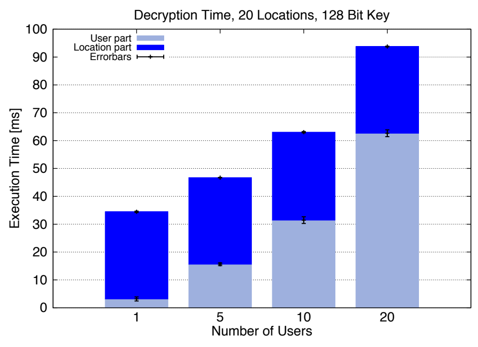
In Figure 5.3 we can see that the overhead which is added by encrypting for more locations is constant with an increasing number of users. As Figure 5.3a shows, with 128 bit keys, encryption for one user takes 3.34ms and encryption for one location takes 1.69ms (resulting in a total encryption time of 5.03ms as mentioned above). Encrypting for 20 users however takes 62ms whereas encryption for one location still only attributes to 1.66ms (around 64ms in total).
When we encrypt for 20 locations as seen in Figure 5.3b, we see a similar picture. The user part when encrypting for one user is around the same

1 User	User portion	Location portion	Total
1 Location	3.34 ms	1.69 ms	5.03 ms
5 Locations	3.10 ms	7.91 ms	11.01 ms
10 Locations	3.10 ms	15.63 ms	18.73 ms
20 Locations	3.15 ms	31.30 ms	34.45 ms
5 Users			
1 Location	15.72 ms	1.63 ms	17.35 ms
5 Locations	15.49 ms	7.98 ms	23.47 ms
10 Locations	15.40 ms	15.60 ms	31.0 ms
20 Locations	15.63 ms	31.08 ms	36.71 ms
10 Users			
1 Location	31.28 ms	1.87 ms	33.15ms
5 Locations	31.04 ms	7.88 ms	39.92 ms
10 Locations	31.06 ms	15.82 ms	46.88 ms
20 Locations	31.46 ms	31.58 ms	63.04 ms
20 Users			
1 Location	62.09 ms	1.66 ms	63.75 ms
5 Locations	62.09 ms	7.76 ms	69.85 ms
10 Locations	63.03 ms	15.67 ms	78.70 ms
20 Locations	62.65 ms	31.13 ms	93.78 ms

Table 5.3: Duration of encryption for different number of users and locations and 128 bit keys

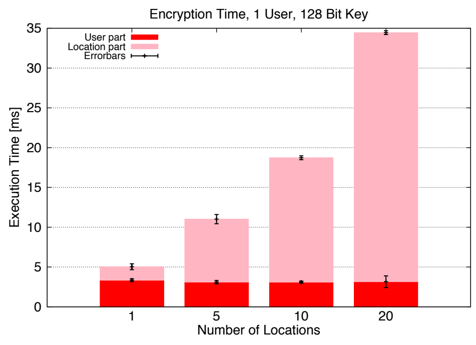


(a) Encrypt for one location

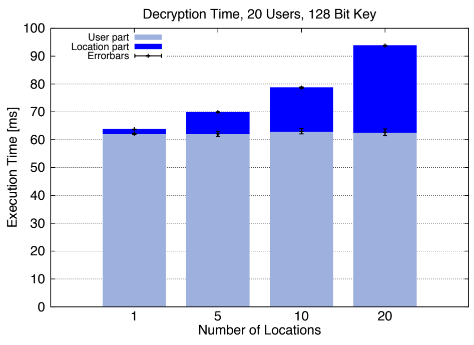


(b) Encrypt for 20 locations

Figure 5.3: Time used to Encrypt for different number of users



(a) Encrypt for one user



(b) Encrypt for 20 users

Figure 5.4: Time used to encrypt for different number of locations

value as before: 3.15ms whereas encrypting for 20 locations accounts for 31.30ms (a total of around 34ms). When the number of users is changed to 20 however, we need 63ms to encrypt for the users and still 31.13 for 20 locations (resulting in about 94ms in total).

Similarly, Figure 5.4 shows that the overhead which is added when we encrypt for different number of users is constant, even if the number of locations we encrypt for changes.
In Figure 5.4a we see again the encryption for one user and one location to be about 5ms. When encrypting for 20 locations we have the same situation as the far left bar in the plot of Figure 5.3b which shows us a total of about 34ms.
Figure 5.4b compares the encryption time needed for 20 users to different number of locations. When encryption is done for one location and 20 users, the result is the same as the one on the far right in Figure 5.3a, which is around 64ms.
Changing the number of locations to 20, the encryption time accounts to about 94ms in total as can be seen on the far right of Figure 5.3b as well. The plots further show that the overhead added by additional locations is slightly less than the overhead added by additional users.

5.3 Analysis

To find out why there is a difference in encryption and decryption times measured, we repeated some of the experiments with a time profiler. The profiler has spent most of the time in the encryption and decryption functions in the function named `BN_mod_exp()`: 83% in case of tag-based encryption, 95% in ElGamal encryption, 43% in tag-based decryption and

38% in ElGamal decryption. This function, originating from the openSSL library, calculates exponentiation modulo a big integer (in our case N). We can already see, that the time spent in this function is a lot larger when encrypting. The main difference between the encryption and decryption functions concerning `BN_mod_exp()` is, that when encrypting, it is called twice, but when decrypting, it is only called once. With this function being the most time-intensive function, we see why encryption takes longer than decryption.

By looking at the decryption plot in Figure 5.1b and the results in Table 5.1 we notice that the location part of the decryption does rise nearly to the same extent than the user portion. The time the user needs to decrypt when using 1024 bit keys is about 60 times as much as a user needs to decrypt when using 128 bit keys. The location part of the decryption is handled about 50 times slower. Therefore we assume that the openSSL library and the python implementation scale similarly.

When directly comparing the time needed for encryption of the location part and decryption by the location server however, we notice that the client is considerably slower for the same key size than the server. This may be due to the fact that the client runs on an iPad which has considerably less computing power than the MacBook Pro which runs the server.

Comparing the absolute time needed to encrypt in Figure 5.1 to the plots in Figures 5.4 and 5.3 suggests that the key size is the most important factor, i.e. has more influence on the encryption time than the number of users or number of locations.

The fact that encryption time increases by a constant factor for varying number of users/locations (see Figures 5.3 and 5.4) suggests that the two factors "number of users" and "number of locations" are independent of each other.

Chapter 6

Related Work

We combine sharing files in the cloud with location-based access control and anonymity of users. Related work in these three fields is presented in this chapter.
First, an introduction to other privacy preserving techniques is given. For every technique we argue why we did not use it to introduce privacy in LoTAC. In Section 6.2 some models which achieve location-based access control are explained. At last, some solutions for achieving location privacy are investigated.

6.1 Privacy Preserving Techniques

We investigated two different types of privacy preserving techniques in depth: anonymous credentials and dynamic accumulators. They are presented in the two Sections below.

6.1.1 Anonymous Credentials

Overview Anonymous credentials as introduced e.g. by Lysyanskaya [19], Baldimtsi [5] and Camenisch[10] are based on the idea that an authorized user interacts anonymously with different organizations.
It is assumed that there is some sort of certification authority (CA) which authenticates the user. To be able to interact with other organizations, the user establishes a pseudonym with these organizations.
When transferring information back and forth between the organizations, this is done using a so-called credential. An organization may issue a credential for a certain pseudonym (or simply "nym"). The user may then

transfer this credential to another nym he has with another organization. This could be used to e.g. get a doctor's note (first organization) and "transfer" it to the employer (second organization). The doctor and employer would then not be able to match their employee and patient, but the employee would be able to show a valid doctor's note to his employer.

Application to LoTAC The explained model is not easily applicable to LoTAC.
First, to be able to authenticate a user and make sure a user of the system corresponds to exactly one person, a certification authority is needed. LoTAC does not have a central point of authentication. Therefore, we would need to newly introduce a CA into our system, which would counteract our goal of a lightweight, additional layer.
A second problem is that we want users to be completely anonymous in respect to their requests at different location servers. This includes that a single location server should not be able to distinguish between a request from the same or one from a different user. Since with anonymous credentials, a user establishes a pseudonym with an organization and re-uses this pseudonym, our requirement would have made it necessary to establish a new nym with every request a user makes which would have generated a huge overhead. Furthermore, the location servers need the master public key of a user for reasons explained in detail in chapter 2. Since only a CA knows a users' master public key, some kind of credential transfer would be needed. This would imply that the user needs to get a credential from the CA for every interaction with a location server which would generate a considerable overhead as well.

Implementations Below, three implementations of the above introduced model are further explained.

Lysyanskaya, Rivest and Wolf [19] propose a system which allows single-use and multiple-use credentials. Single-use credentials are similar to electronic coins in the sense that double-spending enables organizations to link different nyms of the user. The public- and private keys of the user are private input to the nym generation protocol which uses zero-knowledge proofs, i.e. the organization does not know the user's public key, therefore the user stays anonymous.
Their CA is only necessary as an incentive for users not to share their identity. However, LoTAC does require users to keep their secret key private, so a CA would be necessary.
Their construction is as follows: A user has a public key g^x where x is the

private key. To generate a nym, they first blind the public key by choosing a random parameter γ and raise $g^{\gamma x}$. Then, the nym is formed by choosing another random parameter r: $(g^{\gamma r}, g^{\gamma x r})$. To prove that the nym and the master public key are derived from the same private key x, they use a protocol by Chaum and Pedersen [13] which proofs equality of discrete logarithms.
To issue a credential, they blind the transcript of the above mentioned protocol by Chaum and Pedersen such that not even the prover may tell to which conversation it belongs. These blinded transcripts can then be used to transfer a credential because they do not leak the user's pseudonym with the issuing organization, nor the issuing organization but prove that a user is the one he claims to be.

In a later work [10], Camenisch and Lysyanskaya enhance the model with anonymity revocation, which can be used to prevent misuse of anonymity. They, too, differ between one-show and multiple-show credentials. They further introduce a property referred to as "all-or-nothing-sharing". A user is then not able to share a pseudonym with someone else without handing over access to all of his secret information as well.

A more recent idea of Baldimtsi and Lysyanskaya [5] uses blind signatures and pedersen commitments to construct an anonymous credential system. A credential consists of a blinded transcript of a proof of knowledge where a user proofs that he committed to the correct attributes (e.g. his identity). The signer himself is not going to be able to distinguish what he signed from other credentials because an OR-proof technique is used. This technique works by running two proofs of knowledge (more precise: Σ-protocols) at the same time in such a way that that the signer does not know afterwards which of the two statements was proven.

6.1.2 Dynamic Accumulators

Overview The idea of accumulators as introduced by Benaloh and de Mare [6] is to accumulate a large set of inputs into a short value while being able to prove that a certain value is part of the set. Camenisch and Lysyanskaya [11] propose a dynamic accumulator to which values can be added and removed dynamically. Proving that a value is part of the accumulated set is done by a proof of knowledge in which a user has to prove two things: first, he did commit to a value and second, this same value is part of the accumulator. The verifier must not learn the value during the proof of knowledge. The proof must not succeed if the value is not part of the accumulator.

Construction The base of an accumulator is a set of numbers which are accumulated by an accumulation function. The function takes as arguments the old accumulator and the new value to be added. The result of this function is called the new accumulator. In addition, there is the notion of a witness. Witnesses are updated using the same accumulation function by taking as arguments the old witness and the new value to be accumulated.

Identity Escrow Scheme Camenisch and Lysanskaya further propose an identity escrow scheme based on dynamic accumulators.
The initial values for the accumulator and the witness are chosen randomly by the group manager. Joins and revocations of membership as introduced above are handled by the group manager as well.

Proving that a value x was incorporated into the accumulator with witness w is done by checking whether the new witness (i.e. the witness after adding the value to the accumulator) equals to the result of the accumulation function with arguments x and w. The proof uses a zero-knowledge protocol. This requires an update every time a value is added or deleted from the accumulator.

Application to LoTAC Accumulators may be used to prove membership. In LoTAC, there are two memberships a user can have: The membership to the system as a whole (i.e. as a valid user), and the membership to the set of users who may decrypt a certain file ("access set").
It is not needed to prove membership to the system as a whole, i.e. to authorize a user. In the original LoTAC protocol, the authorization functionality of GSM is only used to get hold of the correct public key of a user. When introducing privacy, we instead prove knowledge of the secret key since we only need to make sure that a user uses his own public key but not whether he is a legitimate user of the system. If he is not, he will simply not be able to decrypt the file.
It is not needed to prove membership to the access set either. This could be done either at the cloud server or at the location server, but both is not wanted. Both servers should not need to know about the access policy of a file and are therefore not trusted to enforce it. Introducing accumulators at this point would only add more trust assumptions to the system without gaining any security.

6.2 Location based Access Control

The idea of LoTAC to grant access control based on the location of the user is not new per se. Below, we give an overview over different implementations of the same idea. However, LoTAC is the first practical implementation which does not need to trust the user to report the correct location.

Ray and Kumar [25] developed LRBAC, a location-aware role-based access control model. They provide a formal model for location-based access control by enhancing the Bell-Lapadula model by location information. They do not provide a practical implementation nor do they specify how the location of the user is assessed.

Ray and Kumar [24] further wrote an article in which they extended the MAC (mandatory access control) model to incorporate location information. They propose that a user's location may be assessed either by GPS or by using infrared-sensors (if indoor). Again they do not provide a practical implementation but rather only formalize the model.

Ardagna and Cremonini [2] model an access control engine which takes into account the users' location when granting access. They do not provide an implementation but only describe the mechanism. The biggest difference to LoTAC is, that in LoTAC, the location infrastructure is independent where as in the LBAC (location-based access control) model described by Ardagna and Cremonini, the system consists of a central engine which grants or denies access and which processes location information of the user in order to decide.
As a location mechanism, they propose to use GSM/3G-technologies because of their widespread usage. However, they say that GPS (outdoor) or WiFi (indoor) could be used as well, but do not mention the weakness of relying on user-reported locations such as GPS.
The main issue they concentrate on is, that a location assessment is never 100% certain and therefore a confidence measure needs to be used in order to decide between access granted or access denied.

Further, there is a US Patent [26] filed for location based database access. It does not go into specifics and - as expected by a patent - provides merely an idea instead of a thought-out system. It does however specify that a user's location is assessed using GPS. The patent's idea therefore relies on the client reporting the correct location to the server.

6.3 Location privacy

LoTAC preserves the privacy of a user's location by anonymizing the identity of a user when his location is assessed. This way, not even the location assessing authority knows a user's location but still, the correctness of the assessed location does not depend on the amount of trust we put into the user. The idea of keeping a user's location private, has been discussed in literature on various occasions. We describe some of them here.

Bettini, Wang and Jajodia [8]. investigate privacy issues in location-based services. They define a framework which may be used for risk-evaluation and propose ideas which may prevent a user's identification based on his location information. Their model consists of a trusted server which handles sensitive user information such as identity and location. Requests from the user to the service provider as well as responses from the service provider to the user get routed through this trusted server. The service request a service provider receives from the trusted server only contains a pseudonym (which allows the necessary authorization though), an area and a time interval. Neither the exact location nor the exact time of a user's request are handed over to the service provider.

In a later work, Bettini, Wang, Mascetti and Jajodia [7] further extend this model and formally define a framework to guarantee anonymity in location-based services. They build their model on the above mentioned idea of a trusted server which knows the precise locations of each user and handles requests from these users to the service provider. The model is extended by the idea that an adversary could attack a user's anonymity by analyzing masked requests from the trusted server to the service provider. They provide a so-called defense function which guarantees that the masked request always matches a minimal number of eligible users which renders attacks more complicated. They are working on an extension which defends against attacks based on multiple traced requests.
Their idea differs to LoTAC in the way that they define a trusted server which knows a user's location and identity and forwards a blurred location and masked identity. LoTAC does not need to trust the location assessment infrastructure.

Ardagna, Cremonini, Damiano et al. [4] use obfuscated techniques to implement user's privacy preferences. User privacy is measured in terms of location accuracy. The more accurate a user's location, the less privacy. Obfuscation works by shifting the center and/or changing the radius of a user's location. However, if the user applies these techniques himself, then whomever he sends his location needs to trust the user to report his location correctly. On the other hand, if e.g. some location assessment server queries a user's location and applies obfuscation, then he needs to be trusted to keep the real location secret. Both of these limitations do not apply to LoTAC.

In [3], Ardagna, Cremonini et al. defined the same idea more formal and added an adversary model. They differ between two issues: First, after an attempted de-obfuscation, the adversary ends up with a more accurate location than the obfuscated one. Second, the adversary may decide whether he ended up with a more accurate location or not. Everyone who does not know the original location is considered as a potential adversary. They do not consider an adversary who traces multiple requests and gains information from comparing them.
They further conducted experiments where they investigated the rate of successful de-obfuscations of an adversary. A success is defined as a de-obfuscation which has "greater relevance" than the obfuscated location. The success rate varies greatly with the degree of manipulation and ranges everywhere between less than 10% and nearly 100%.

Jiang, Wang and Hu [17] analyze location privacy in wireless networks, specifically 802.11 WLAN networks. They differ between two types of adversaries: Third parties who only observe traffic (i.e. sniffers) to localize and expose users. Such an attacker could e.g. be a government. Second, the service provider itself could act as an attacker in certain cases, e.g. by leaking information about a user or deliberately adjusting a base stations' transmission power to gain knowledge about a user. They propose basically three measurements to improve a user's privacy: often changing pseudonyms, silent periods which allow the user to "mix in" with other users and less precision in localization.

Minch [21] made a survey of privacy issues in location-aware mobile devices. He counts thirteen issues, among them are legal topics and questions of how much the user should be allowed to decide. He futher talks about abusive use of collected location information and identifiability/anonymity of users, though he does not propose a solution.

Chapter 7

Conclusion and Future Work

This last chapter summarizes our original goal and our contributions. In Section 7.2 we provide some ideas for future research on this project.

7.1 Conclusion

With files being saved in the cloud more and more often, they can be accessed from any location at any time. This requires not only sophisticated access control mechanisms, but enables a lot of possibilities for data collection as well as privacy violations. LoTAC provides location and time-based access control for cloud-stored data without the need for a file owner to trust the cloud server or the location assessing servers nor the users. On top of this system, we implement the ability for a user to stay anonymous while accessing files. With our extension to LoTAC, access control, i.e., authorization, is left intact while the user accessing a file stays unidentified. We reach our goal by making use of techniques borrowed from blind signatures and zero-knowledge proofs.

In addition, we implement LoTAC for the iPad and show that its performance is very satisfying even on a mobile device. We measured the time used to encrypt and decrypt and show that both take only a few milliseconds with 128 bit keys. Furthermore, encrypting for more than one user and more than one location does only add linear overhead.

In summary, we extended LoTAC, which achieves location and time-based access control, with a feature providing user privacy. Moreover, we implemented LoTAC for the iPad and showed its adequacy for mobile devices with performance measurements.

7.2 Future Work

Anonymous Files

We introduced anonymity of users into LoTAC. Files however may still be traced by colluding location servers since the ciphertext a user uploads to the location server is unique per file. If a file is not requested by a lot of users, then this may enable colluding location servers to trace the journey of users. Even if it is not possible to identify the users, tracing a user is not desirable either, therefore it would be most interesting to introduce anonymity for files as well.

Anonymous Policy

The time tags are submitted in plain text to the location server. Since files are traceable, it is possible for colluding location servers to reconstruct the policies of files. Although a location server may not know whether a submitted time tag is valid, if a file is requested by a lot of users and therefore a time tag is submitted often, the location server may conclude that the time tag is valid.
Reconstructing policies opens a lot of interesting possibilities for data collection. Location servers may gather information about the policies most often used (e.g., weekdays during office hours) and based on that e.g. adapt their computing power. However, from the view of an owner, it is not desirable at all that anyone else than a user leans about the policy. The owner may e.g. be a financial corporation where it is very important that no one knows about certain points in time because it could for example influence stock prices.
Furthermore, not only as an owner but as a user as well it may be undesirable that information about policies is collected. The location servers may gather unfavorable statistics, for example, that the users tend to request a file at the end of the permitted time period or tend to go first where time ranges are longer. If a file has a lot of users then it would of course not be possible anymore to determine where a user has been first because users are indistinguishable.
But if files would be untraceable then it would be impossible to reconstruct a file's complete policy and therefore this problem would be much smaller. It would however still be possible for a single location server to gain knowledge about supposedly permitted time frames for his own location.

Stationary Decryption

Once a user has obtained the key to decrypt a file, he may decrypt that file whenever and wherever he wants. This is of course not always desirable. If the owner of a file is a company which wants the employees to access a file only if they are withing the company's buildings or if the company wants to ensure, that, e.g., the employees do not take the file home to read, then LoTAC is not able to provide the necessary solution. An extension which would make it possible that files may not be read outside the locations and times specified in the policy would be a big improvement. Such an extension, however, is not trivial. A user would not only have to forget the decrypted key, we would also need to make sure that he can not save the decrypted file in any way.

Confirmation of access

An owner never learns if a certain user has accessed a file and if the user has accessed it, whether the attempt was successful. This ability may however be desired for different reasons, e.g., an owner would want to make sure that everyone which is part of a project has read certain guidelines. It may be interesting for features like billing a user for accessing a file as well.

Bibliography

[1] Elli Androulaki, Claudio Soriente, Luka Malisa, and Srdjan Čapkun. Enforcing location and time-based access control to cloud-stored data. unpublished.

[2] Claudio Ardagna and Marco Cremonini. Supporting Location-Based Conditions in Access Control Policies. *Proceedings of the 2006 ACM Symposium on Information, computer and communications*, pages 212–222, 2006.

[3] Claudio Ardagna, Marco Cremonini, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An Obfuscation-Based Approach for Protecting Location Privacy. *IEEE Transactions on Dependable and Secure Computing*, 8(1):13–27, January 2011.

[4] Claudio A. Ardagna, Marco Cremonini, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. Location Privacy Protection Through Obfuscation-Based Techniques. *Data and Pllications Security 2007*, pages 47–60, 2007.

[5] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. *IACR Cryptology ePrint Archive*, 2012.

[6] Josh Benaloh and Michael De Mare. A Decentralized Alternative to Digital Signatures (Extended Abstract). *Advances in Cryptology EUROCRYPT 93*, pages 274–285, 1994.

[7] Claudio Bettini, Sergio Mascetti, X. Sean Wang, and Sushil Jajodia. Anonymity in Location-Based Services: Towards a General Framework.

2007 International Conference on Mobile Data Management, pages 69–76, May 2007.

[8] Claudio Bettini, Xiaoyang Sean Wang, and Sushil Jajodia. Protecting Privacy Against Location-Based Personal Identification. *SDM*, pages 185–199, 2005.

[9] Dallas Brown and Eric Czarny. Xml-rpc client for ios. https://bitbucket.org/kdbdallas/xmlrpc-ios, June 2012.

[10] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. 2001.

[11] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO'02*, pages 61–76, 2002.

[12] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology Proceedings of Crypto*, volume 82, pages 199–203, 1983.

[13] David Chaum and Torben P. Pedersen. Wallet Databases with Observers. In *Advances in Cryptology Crypto 92*, volume 740 of LNC, pages 89–105, 1992.

[14] Shafi Goldwasser, S Micali, and C Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[15] Martin Hirt and Ueli Maurer. *Cryptographic Protocols Lecture Notes.* 2010.

[16] Dennis Hofheinz and Eike Kiltz. The Group of Signed Quadratic Residues and Applications. *Advances in CryptologyCRYPTO 2009*, 5677:637–653, 2009.

[17] Tao Jiang, Helen J. Wang, and Yih-Chun Hu. Preserving Location Privacy in Wireless LANs. *MobiSys'07*, pages 246–257, 2007.

[18] Stefan Lucks. A variant of the cramer-shoup cryptosystem for groups with unknwon order. *IACR Cryptology ePrint Archive*, pages 52–52, 2002.

[19] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography'99*, pages 184–199, 1999.

[20] Philip Mackenzie, Michael K. Reiter, and Ke Yang. Alternatives to non-malleability: Definitions , constructions , and applications (extended abstract). *Theory of Cryptography*, pages 171–190, 2004.

[21] Robert P. Minch. Privacy Issues in Location-Aware Mobile Devices. *Proceedings of the 37th Hawaii International Conference on System Sciences*, pages 1–10, 2004.

[22] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in CryptologyCRYPTO'92*, 1992.

[23] The openSSL project and Eric Young. openssl toolkit. `http://www.openssl.org/`, February 2013. Version 1.0.1e.

[24] Indrakshi Ray and Mahendra Kumar. Towards a location-based mandatory access control model. *Computers & Security*, 25(1):36–44, February 2006.

[25] Indrakshi Ray, Mahendra Kumar, and Lijun Yu. LRBAC : A Location-Aware Role-Based Access Control Model. *Lecture Notes in Computer Science*, 4332:147–161, 2006.

[26] Melvin J. Teare and Stephen S. Walker. United states patent number 5'243'652: Location-sensitive remote database access control, September 1993.

[27] Michael Tyson and Stephen Lombardo. Xcode project skeleton for openssl. `https://github.com/michaeltyson/openssl-xcode`, 2012.

List of Figures

List of Tables