UNIVERSITY OF PIRAEUS
Department of digital systems
Postgraduate program "Digital Systems Security"

Master Thesis

Himitsu Project The Beginning: Initial Architecture and Multiparty Support

Dimitrios Desyllas MTE-1710

supervised by
Dr. Lamprinoudakis Konstantinos

March 19, 2019

Contents

Abstract

On my previous assay I have studied various Privacy Enhancing Technologies [1], but these offered privacy on only situations where a user needed to communicate with an other user. Furthermore in it there were huge user experience gaps as well. As a result I proposed the Himitsu project, as an attempt to offer privacy with awesome user experience. So in this essay I go a step further and study technologies for multiparty Instant Messaging and group key agreements used for this purpose. Afterwards I propose the initial Himitsu architecture based on knowledge acquired from the study of the technologies above.

Chapter 1

Περίληψη στην Ελληνική Γλώσσα

Στην εργασία αυτή γίνετε μια μελέτη στο πως προστατεύεται η ιδιωτικότητα σε συστήματα ανταλλαγής στιγμιαίων μηνυμάτων πολλών προς πολλούς. Ο απώτερος σκοπός η δημιουργία μιας αρχικής αρχιτεκτονικής του μακροχρόνιου project με την ονομασία Himitsu, ενός συστήματος που επιτρέπει στους τελικούς χρήστες να δημοσιεύουν στους φίλους τους υπηρεσίες και μέσω αυτών να ανταλλάσουν δεδομένα. Το βασικό εργαλείο που αξιοποιείτε για την προστασία της ιδιωτικότητας είναι η άκρο-προς-άκρο κρυπτογραφημένη ανταλλαγή δεδομένων (end-to-end encryption).

Στην αρχή μελετάμε με ποιους αλγορίθμους μπορεί να επιτευχθεί ενός κεντρικού ομαδικού κλειδιού, που αργότερα μπορεί να αξιοποιηθεί για την ανταλλαγή δεδομένων. Μελετάμε την λειτουργία τους καθώς και μοντέλα ανάλυσης ασφάλειας καθώς και τα υπολογιστικώς δύσκολα προβλήματα στα οποία η ασφάλεια αυτών βασίζεται. Ακόμη δε, βλέπουμε ότι οι αλγόριθμοι βασίζονται είτε σε αρχιτεκτονική κύκλου στην οποία αξιοποιείτε ο προηγούμενος συμμετέχοντας και ο επόμενος στον σχηματισμό ενός ενδιάμεσου κλειδιού, αργότερα συνδυάζονται όλα τα ενδιάμεσα κλειδιά στην δημιουργία ενός ενιαίου κλειδιού. Μια εναλλακτική προσέγγιση είναι η χρήση δέντρου στο οποίο ο συνδυασμός φύλλων κλειδιών έως την ρίζα δημιουργεί ένα ενιαίο κοινό συμμετρικό κλειδί.

Αργότερα βλέπουμε τρόπους που διαδεδομένες εφαρμογές και αλγόριθμοι εφαρμόζουν μηχανισμούς ιδιωτικότητας στην αρχή βλέπουμε το (N+1)sec και το mPOTR που, εφόσον δημιουργηθεί ένα κοινό συμμετρικό κλειδί, πέρα από την εμπιστευτικότητα με την άκρο-προς-άκρο κρυπτογράφηση δεδομένων προσφέρει και άλλους μηχανισμούς προστασίας όπως ή διαμφησβιτισιμότητα (deniability), ή συνοχή μηνυμάτων (consesus) και άλλους. Τέλος βλέπουμε και άλλες εφαρμογές όπου εφαρμόζουν έναν δικό τους τρόπο χωρίς όμως να εφαρμόζουν στο σύνολο τους όσες υπηρεσίες προσφέρει το (N+1)sec και mPOTR.

Μετά κάνω 2 απόπειρες ανάπτυξης γνωστών αλγόριθμων συμφωνίας κλειδιών που αναφέρω στο κεφάλαιο 3. Η πρώτη είναι υλοποίηση του αλγορίθμου mDP+P σε Javascript με την χρήση electron και του πρωτοκόλλου επικοινωνίας XMPP και η δεύτερη είναι αξιοποιώντας C και MPI σαν μηχανισμό ανταλλαγής δεδομένων προκειμένου να υλοποιήσω το πρωτόκολλο Burmester Desmedt.

Επιπλέων, βάση των γνώσεων και εμπειριών που απέκτησα από τα προηγούμενα κεφάλαια ορίζω μια βασική αρχιτεκτονική για το Himitsu project. Αρχικά ορίζω τα βασικά δομικά στοιχεία του project και αργότερα σε κάθε δομικό στοιχείο ορίζω τις τεχνικές απαιτήσεις αυτού. Μετά ορίζω απαιτήσεις για την διαχείριση των κλειδιών καθώς και τους κρυπτογραφικούς αλγορίθμους οι οποίοι θα αξιοποιηθούν. Τέλος ορίζω βασικά συμπεράσματα από την εμπειρία καθώς και τι ελλείψεις υπάρχουν στην βιβλιογραφία. Τέλος ορίζω ποια θα είναι τα επόμενα βήματα ανάπτυξης του Himitsu project.

Chapter 2

Introduction

2.1 Introduction

Nowdays the landscape of Internet has drastically changed, as a result various problems happened towards user's privacy and freedom of speech. Many offered solutions as seen in [1] lack of the necessary user experience in order for a widespread solution, easy for the average Joe to adopt. As a result I propose the Himitsu project, a long-term project aiming to offer privacy through the end-to-end encryption and the use of user-published services with awesome user experience, via extending the technologies of Instant Messaging.

Initially there will be a bibliographic research regarding the technologies used for Group Key Agreement, the algorithms with focus on ones that used in Instant messaging protocols. There will be analyzed the hard problems and methodologies used for proving their security and on how they works as well. Furthermore we also study the logical overlay communication structure used for message exchanging and based on that we classify the key agreement protocol.

Afterwards a further bibliographic research is performed on existing Instant Messaging protocols focusing on how the privacy is protected, specifically is studied what privacy requirements needs to be met on a multiparty instant messaging system. Also is focused on what cryptographic primitives are used to fulfill the privacy requirements and how effectively are being used.

Next, attempts are described on implementing a group key agreement. There were 2 attempts one Implementing the mDP algorithm using electron and XMPP and one implementing Burmester Desmedt using C and MPI. At the end the Himitsu's initial architecture, requirements and technologies used are being described.

Having studied all the protocols mentioned above, I achieved to make the initial architecture for the Himitsu Project and define its initial technical and security requirements, based upon existing technologies and architectures used for instant messaging, even though I failed to implement a group key agreement. Also I managed to define a development roadmap and I figured out what necessary steps are required in order to develop the required components (either protocol-based ones or the software-based ones) of it.

2.2 Goals of the essay

The goals of the assay are the following:

- To study existing software applications and protocols used for key agreement and end-to-end encryption upon instant messaging.
- Study possible libraries and protocols as candidates for Himitu's protocol stack.
- Define the initial technical requirements and overlay architecture for the Himitsu Project [1].

The study has been focused on group instant messaging only, thus key agreements or any other technology for group key agreement or group data exchange has been rejected.

2.3 Hypothesis

I assume that there are a group of n participants that share a common message exchange channel. All participants have common rights upon the channel (there are no privileged users) and the owner of the communication channel is considered malicious and does not participate to the data exchange upon the channel. A user of this channel broadcasts a message towards to other users and thus will be mentioned as *data exchange*.

The malicious owner of the channel is able to:

- Analyze the traffic and target users for various reasons (eg. commercial, political etc etc)
- Corrupt exchanged messages either to cause a denial of service or for other reasons.
- Redirect the data exchange to another parties.

Also is assumed that the owner of communication channel took any measures against 3rd party unauthorized access and the only malicious users that is able to have this channel are:

- A subset of m ($1 <= m < n$) users that participate in the data exchange.
- The communication channel owner himself/herself, or other parties that legally he/she gives access to it.

Furthermore the communication channel ensures that no message will be lost and all messages will be successfully be transfered as well. The only message corruptions are the intentional from malicious users in a *data exchange* session, the channel owner and parties that have granted access to it either via the owner of via a channel compromise via malicious actions towards it.

Chapter 3

Analysis of existing bibliography the Group Key Agreements used in existing instant messaging protocols.

3.1 Introduction

A way to achieve end-to-end encryption is for the participants of the channel to agree upon a symmetrical key in a similar manner as Diffie Hellman protocol does. Afterwards can be used either stream of block cyphers to protect the confidentiality of the messages. In a privacy - enhanced group key agreement is used during the *setup* phase. Some of them go beyond establishing a group key and offer guarantees for deniability (which will be explained on chapter 4) as well.

3.2 Hard problems and security models that group key agreements rely upon.

3.2.1 Diffie Hellman problems.

Mostly the Group Key agreement extend and use the Diffie Hellman algorithm [2]. Thus their security rely upon the difficulty of these problems:

- **Computational Diffie Hellman Problem [2, 3]**: Given $a = g^x \mod p$, $b = g^y \mod p$ and provided p, g is computationally hard for a polynomial adversary A to calculate $c = a^{\log_g(y)} \mod p$.
- **Decisional Diffie Hellman problem [2, 3]**: Given the following amounts $a = g^x$,$b = g^y$,$c = g^{x*y}$. For a polynomial adversary A it should be computationally hard to find out whether $c = g^{x*y}$.
- **Squaring Computational Diffie Hellman [3–5]**: From a given p, g, a is hard for a polynomial adversary A computationally to find $x^{log_g(x)} \mod p$ and if $x = g^a \mod p$ is also computationally hard to find whether an amount $c = g^{a^2} \mod p$.
- **The squaring Decisional Diffie-Hellman [3, 6]**: Given p, g and 2 random numbers x, z it is computationally hard for a polynomial adversary A to decide whether $z = x^{log_g(x)} \mod p$.

3.2.2 CK and eCK security Models.

Furthermore there is a model where allows us to check whether a group key agreement is rather secure by standardizing adversary's strengths and actions. The models used in the bibliography are either similar or exact the same with the Canetti-Krawczyk [7] and the extended version by LaMacchia Lauter and Mityagin [8].

In the CK model [7] is a model developed by Ran Canetti and Hugo Krawczyk, defines that an attacker is able to perform the following actions [7]:

- **Session State reveal** Depending the implementation of the key agreement protocol the adversary is able to obtain information of a local state of the protocol under an incomplete session eg. secret keys generated on a participant's client.
- **Session-key query** The participant is able to obtain the key(s) of a completed key agreement session.
- **Party Corruption** Where at any point of the key agreement the attacker is able to impersonate an another user by obtaining a participant's secret amounts.
- **Session Expiration** The attacker is able to terminate a session using protocol's messages are broadcasted to the session or sent directly to one participant. Thus the purpose is to make the participant(s) to think that the session has been terminated.
- **Bootstrapping the security of key-exchange protocols.** A key agreement protocol requires some cryptographic parameters such as common amounts for calculations eg. Diffie Hellman's p and g parameters. The attacker is able to modify these parameters and initialize the key agreement protocol.

Whilst at the eCK [8] the CK Model [7] is extended in modern key agreement protocols each participants have at least 2 keys, a long term key and an ephemeral one, for example in the case of using the Triple Diffie Hellman Key Agreement [9]. Also at the standard CK [7] does not allow to perform the following attacks where the attacker reveals long-term or ephemeral keys of ate least one party and then impersonates him/her [8].

So with the extended CK model the adversary is also allowed to performs these attacks as well [8]:

- **Ephemeral Key reveal** Where the adversary is able to reveal the ephemeral key of a party and replaces the Session State reveal attack from standard CK [**CK**] model.
- **Long-Term Key Reveal** Where the adversary is able to reveal the Long-term key of a party without corrupting the party.

With these actions above the adversary is able to do a party corruption thus there's no definition for party corruption in this model.

3.3 Top Level Hierarchy for Group Key Agreements

The participants of a group key agreement are placed upon a overlay structure. The structure can be either a tree where each participant is placed as a leaf of it and the group symmetrical key is the root of it. Furthermore there are intermediate nodes by combining n-children. The protocols using this approach are the Asynchronous Ratcheting Tree [10], TreeKem [11].

Another approach is to place the participants on top of an overlay ring where the participant during the key agreement execution can broadcast its public key. An overlay approach on these types of key agreements is the following:

1. Each participant broadcasts its public key.
2. Using previous and next neighbor calculate an intermediate value and broadcast it.

3. Using these values calculate the common key.

This approach is common in the classical Burmester Desmedt Protocol [12, 13] and in the mDP+P/mdp+S protocol [14].

3.4 Protocols for key agreement used in Instant messaging

3.4.1 Burmester Desmedt protocol

At 1995 Mike Burmester and Yvo Desmedt [12] created a protocol for group key agreement that extends the classical Diffie Hellman protocol for generating a group key for n-sized communications where $n > 2$. In order to achieve that follows the following steps [12, 13]:

1. Let suppose we have a n-sized party containing the following participants $[U_1, U_2, \ldots, U_n]$. The U_i is the i-th participant and the participants are considered in a cyclic group meaning: $U_0 = U_n$ and $U_{n+1} = U_1$.
2. Each participant U_i generates a random number x_i as secret key and broadcasts its public key $z_i = g^x \mod p$.
3. When the participant U_i receives the z_{i-1} and z_{i+1} then calculates and broadcasts: $X_i = (z_{i+1}/z_{i-1})_i^x \mod p$.
4. The final key is: $K = z_{i-1}^{nx_i} X_i^{n-1} X_{i+1}^{n-2} \ldots X_{i-2} \mod p$

As you can see the its security relies upon the hard problem of *Gap Diffie Hellman*. Further armoring is the use of digital signatures where the message integrity and authentication can be done as well with the use of permanent pairs of Public and Private keys alternative approach is the use Tripple Diffie hellman [9, 15] instead of a simple Diffie Hellman, improving privacy and deniability as well.

Key Agreement in GOTR

A intriguing approach and use of Burmester Desmedt protocol is in the GOTR protocol [16] where in each participant U_i of an n sized group has $2n$ virtual nodes, 2 virtual nodes per another connection. When a user joins to a group each participant performs a Burmester Desmedt Key agreement with the other participant nodes in this sequence:

- To the joining participant is performed a Burmester Desmedt on pair with the other's Virtual nodes so for an n-sized group the key agreement is performed n times.
- Then we can re-use the existing virtual nodes to execute yet another Burmester Desmedt key agreement in order to create a session with the other participants.

As you can see at least 1 participant will need to perform twice the Burmester Desmedt key agreement, on the other hand it makes it easier for subgroup creation due to reuse of intermediate keys.

3.4.2 Bohli Key Agreement

At Vietcrypt in 2006, Jean-Mattias Bohli and Rainer Steinwandt has presented a way to perform deniable key agreement [17]. The protocol goes as follows:

1. Each participant generate a random value k_i and a random value x_i calculates $y_i = g^{x_i}$. Then broadcasts: $M_i^1 = (H(k_i), y_i, U_i)$
2. Then upon receival of $M_j^1, 1 \geq j \leq n$ the $sid_i = H(pid_i || H(k_1) || \ldots || H(k_n))$ is calculated. Also a Shorr Blind Signature is calculated by choosing a random integer r_i and calculating $z_i = g^{r_i}$. Afterwards the message $M_i^2 = (sid_i, z_i, U_i)$ is broadcasted.

3. Upon receival of M_i^2 calculate the left and right keys $t_i^L = H(y_{i-1}^{x_i})$ and $t_i^R = H(y_{i+1}^{x_i})$. Then we calculate the $T_i = t_i^L \bigoplus t_i^L$ and broadcast $M_i^3 = (k_i \bigoplus t_i^R, T_i, U_i)$.

4. Upon receival of $M_j^3, 1 \geq j \leq n$:

   - Check whether $T_1 \bigoplus T_2 \dots T_n = 0$ and for all decrypted (via XOR) k_i have $H(k_i) = M_i^1[0]$ where $M_j^1[0], 1 \geq j \leq n$ (where 0 is the first element of the message).
   - Calculate the session key $sk_i = H(pid_i||k_1|| \dots ||k_n)$
   - Do the same for the session confirmation $sconf_i = H((y_1, k_1)|| \dots ||(y_n, k_n))$
   - Hash them together $c_i = H(sid_i||sconf_i) \mod q$
   - Calculate the $d_i = r_i - c_i \alpha_i mod q$.
   - Broadcast: $M_i^4 = (d_i, U_i)$

5. Then Verify the key for received $M_j^4, 1 \geq j \leq n$ via confirming $z_z = g^{d_j}(PK_j)_i^c$.

Where $pid_i = (U_1, U_2, \dots, U_i, U_n)$ for $1 \geq i \leq n$ on an n-sized chatroom and $H()$ a cryptographically secure function. The final key is the sk_i for a participant U_i.

Further addictions have been inserted on the Mathew's proposal [18] for sender authentication via generating a Signature key on the first step of key agreement and using it for origin authentication. So in the last step the message that will be received will be is $M_i^4 = (d_i, U_i, \sigma_i)$ and the confirmation for the key will be $sconf_i = H((y_1, k_1, S_i)|| \dots ||(y_n, k_n, S_n))$ where S_i the signature public key for participant U_i. Also at the last step in order to agree on the key the signatures will need to be verified.

As I notice, with the use of the ephemeral key the element of deniability is added so each participant can deny that has done a encrypted multipaty data exchange session. It is a similar approach used on later implementations of key agreements such as Tripple Diffe Hellman where permanent and ephemeral keys are used for deniability. Also a huge amount of exchanged messages is being done (3 broadcasts per user) instead of the classic Burmester Desmedt [12] and mDP+P [14] protocols (1 or 2 broadcasts per user).

3.4.3 mDB+P protocol

Original protocol

This protocol is attempted to Combine the Parallel Diffie Hellman [14] and Burmester Desmedt protocol mentioned above. It was presented in a Paper from Michel Abdalla, Celine Chevalier, Mark Manulis and David PointCheval at Africacrypt 2010 [14]. The protocol goes as follows:

1. Let suppose we have a n-sized party containing the following participants $[U_1, U_2, \dots, U_n]$. The U_i is the i-th participant and the participants are considered in a cyclic group meaning: $U_0 = U_n$ and $U_{n+1} = U_1$

2. Each participant U_i generates a random number x_i as secret key and broadcasts its public key $y_i = g^x mod p$.

3. Upon receival of all y_i then each U_i computes the following:

   - $sid_i = (U_1|y_1, U_2|y_2, \dots, U_n|y_n)$
   - $k'_{i-1} = y_{i-1}^{x_i}$ and $k'_{i+1} = y_{i+1}^{x_i}$
   - $z'_{i-1,i} = H(k'_{i-1}, sid)$ and $z'_{i+1,i} = H(k'_{i+1}, sid)$ where $H()$ a cryptographically secure function.
   - $z_i = z'_{i-1} \bigoplus z'_{i+1}$
   - $\sigma_i = Sign(sk_i, (U_i, z_i, sid_i))$ where sk_i a signature key for the U_i and $Sign()$ a cryptographically secure signature function.

- Broadcast: (U_i, z_i, sid_i)

4. **Group Key computation**
   - If either each σ_i or $z_1 \bigoplus z_2 \bigoplus \cdots \bigoplus z_n = 0$ abort.
   - For each $j = i, \ldots, i+n-1$ calculate $z'_{j,j+1} = z'_{j-1,j} \bigoplus z_j$
   - The final key is $k_i = H_g(z'1, 2, \ldots, z'n, 1, sid_i)$ where H_g a cryptographically secure function.

Found vulnerabilities

But at the original protocol the Qifeng and Chuangui [19] found the following security weakness. The attack requires a user U_i to be "sandwiched" between 2 malicious users U_{i+1} and U_{i-1}. The malicious users agree to calculate:

- For U_{i-1} : $z_i = z_{i2,i1} \bigoplus z_{i1,i} \bigoplus r_M$
- For U_{i+1} : $z_i = z_{i,i+1} \bigoplus z_{i+1,i+2} \bigoplus r_M$

The r_M mentioned above is a random integer value secretly agreed between U_{i-1} and U_{i+1}. With that the user U_i calculated a different key from the others. As a result either a denial of communication or easier message forgeability from users U_{i-1} and U_{i+1}. In order to fix that the algorithm should proceed as follows:

1. Let suppose we have a n-sized party containing the following participants $[U_1, U_2, \ldots, U_n]$. The U_i is the i-th participant and the participants are considered in a cyclic group meaning: $U_0 = U_n$ and $U_{n+1} = U_1$
2. Each participant U_i generates a random number x_i as secret key and broadcasts its public key $y_i = g^x modp$.
3. Upon receival of all y_i then each U_i computes the following:
   - $sid_i = (U_1|y_1, U_2|y_2, \ldots, U_n|y_n)$
   - $k'_{i-1} = y_{i-1}^{x_i}$ and $k'_{i+1} = y_{i+1}^{x_i}$
   - $z'_{i-1,i} = H(k'_{i-1}, sid)$ and $z'_{i+1,i} = H(k'_{i+1}, sid)$ where $H()$ a cryptographically secure function.
   - $z_i = z'_{i-1} \bigoplus z'_{i+1}$
   - $\sigma_i = Sign(sk_i, (U_i, z_i, sid_i))$ where sk_i a signature key for the U_i and $Sign()$ a cryptographically secure signature function.
   - Broadcast: (U_i, z_i, sid_i)
4. **Group Key computation**
   - If either each σ_i or $z_1 \bigoplus z_2 \bigoplus \cdots \bigoplus z_n = 0$ abort.
   - For each $j = i, \ldots, i+n-1$ calculate $z'_{j,j+1} = z'_{j-1,j} \bigoplus z_j$
   - The final key is $(k_i, k_i^{kc}) = H_g(z'1, 2, \ldots, z'n, 1, sid_i)$ where H_g a cryptographically secure function.
5. **Key Confirmation Message**:
   - Calculate: $M_i = H_k c(k_i^{kc}, sid_i)$
   - Sign: $\sigma_i^{kc} = Sign(U_i, M_i, sid_i)$
   - Broadcast $(U_i, M_i, \sigma_i^{kc})$
6. **Key Verification**: Each U_i verifies the signatures σ_j^{kc} and checks whether $M_j = M_i$ where $j \neq i, 1 \leq j \geq n$.

The idea behind the Qifeng and Chuangui [19] is to hash some common generated value hash and sign it and then confirm whether the value is common for all participants.

Implemented variation on existing project:

At (N+1)sec protocol [15] the protocol is implemented as follows:

1. Let suppose we have a n-sized party containing the following participants $[U_0, U_1, \ldots, U_{n-1}]$. The U_i is the i-th participant and the participants are considered in a cyclic group meaning: $U_0 = U_n$ and $U_{-1} = U_{n-1}$
2. Each participant U_i generates a random number x_i as secret key and broadcasts its public key $y_i = g^x modp$.
3. Then with the use of Triple Diffie Hellman A [9], the user U_i calculates the Triple Diffie Hellman $d_{i-1,i} = TDH(k_{i-1}, y_{i-1}, k_i, y_i)$ and $d_{i,i+1} = TDH(k_i, y_i, k_{i+1}, y_{i+1})$.
4. Then each U_i calculates $z_i = d_{i-1,i} \bigoplus d_{i,i+1}$.
5. Then we linearly combine all the values d and z. Each U_i computes $d_{i+1,i+2} = z_{i+1} \bigoplus d_{i,i+1}, d_{i+2,i+3} = z_{i+2} \bigoplus d_{i+1,i+2}, \ldots$.
6. The common secret is: $S = d_{0,1}|d_{1,2}|\ldots|d_{n-2,n-1}|d_{n-1,0}$. The secret is passed towards a Key derivation function.

As you can see there are the following differences from the original mDP+S algorithm:

- First of all the participants are numbered from $0, \ldots, n-1$ like arrays in a conventional programming language (C, Java, Javascript, C++ etc etc).
- No check whether z_i generated values XOR'ed all together result 0.
- Lack of signatures, the role that signature does (verification that some amount has been generated from a user) is provided by the Triple Diffie Hellman.
- No Key verification steps as Ma and Cheng.

So we can conclude that the changes have been done for easiness on software development. Also it is by far noticeable that the audit via nccgroup at 2017 [20] has no reference for potential weaknesses during group key agreement as well.

3.4.4 Tree-Based Algorithms

The latest trend of the group key agreement is the use of so-called Ratcheting tree. The protocols that use the following approach is the Asynchronous Ratcheting (ART in short) Tree from Facebook Research [1] [10] and the TREE KEM used in IETF's Secure chat protocol MLS [2] [11, 21].

Architecture Overview and cryptographic primitives.

The whole idea behind the Ratchet trees are that tree structures used to calculate the group key, each participant is the lead of a tree and the root of the tree is the common secret key. Each parent has 2 children and the keys of each parent node are generated using the following 2 functions [11]:

- **Derive Child Secret:** Where using the secret of the children each parent calculates the secret of the parent.
- **Generate Public Information:** Where from secret values are generated public values.

In TREE-KEM these functions are defined as:

[1] Sample Source Code written in Java in https://github.com/facebookresearch/asynchronousratchetingtree

[2] Various code implementations written in various languages listed in https://github.com/mlswg/mls-implementations/blob/master/implementation_list.md

```
#Derive Child Secret
parent_secret = Hash(child_secret)
#Generate Public Information
parent_private, parent_public = Derive-Key-Pair(parent_secret)
```

While on Asynchronous Ratcheting Tree can be defined as [10]:

```
#Derive Child Secret
parent_secret = pow(CHILD2_PUBLIC, CHILD1_SECRET) mod p
#Generate Public Information
parent_public = pow(g,parent_secret) mod p
```

As you can see at ART using modular exponentiation we can derive both public and private keys, the ART setup is explained on the figure 3.1.

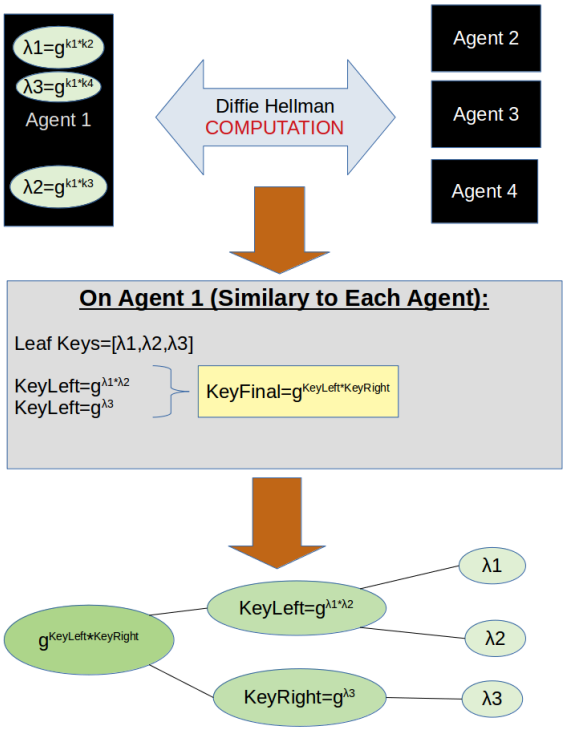


Figure 3.1: How a Asynchronous Ratcheting Tree is being setup

Furthermore the cryptographic primitives in the TREE-Kem algorithm uses the following options in elliptic curves for Diffie Hellman key Agreements [21]:

- Curve25519
- P-256

The cryptographic operations the IETF RFC-5116 AEAD protocol is used [22]. For hashing the SHA-256 used, whilst for data encryption is used the AES-128 in Galois Counter Mode. For key derivation the functions used come from RFC-5869 [21] also the following function is used as well[21]:

```
Derive-Secret(Secret, Label, State) = HKDF-Expand(Secret, HkdfLabel, Hash.length)
```

Tree Structure.

The tree structure used both in ART and TREE KEM is a left balanced one because this can be implemented into an array and with simple mathematical formulas we can easily iterate it. Furthermore comparing ART with TREE-KEM there is some noticeable differences:

- The Tree-KEM uses the latest changes in order to update the tree. Also is allows you according, to the officially released draft [21], to have blank nodes. A functionality that is missing from ART thus it makes really easy on implementations with a small memory footprint.
- ART using the very same function to derive both the public and the private the keys: the modular exponentiation. That means that the a software engineer needs to maintain and side-channel secure only one function the one performing modular exponentiation.
- The test code implementation of the ART protocol the 3 Diffie Hellman is used thus it offers enhanced deniability and privacy compared to Tree-KEM.

Comparison with other protocols mentioned above.

Compared to other protocols it can work with any group size. In case of ART practically a simple Diffie Hellman (and variations of it) is running as well whilst on Tree-KEM the operations are feasible as well with the existing functions. Furthermore according to the researchers ART can handle efficiently an relatively large number of participants 10^3 [21], but there is a lack of a comparison test between the other known protocols used in Instant Messaging. But there are no metrics on Tree Kem also there's a lack of metrics between other key agreement protocols as well.

3.5 Overall Comparison between the protocols

3.5.1 Generic Notices

The first thing I can notice is that there's a lack of any study according to sensitivity in the order that participant join and leave also in modern network applications the messages are received in a no particular order. Practically that means when implemented on asynchronous-friendly technologies like JavaScript and node.js further code is needed for message synchronization. Also at the case of mDP+P and on Burmester Desmedt protocols sometimes a user may not be in a U_i format but for example in case of XMPP may be as *user@example.com/1aad2* so the software developer needs to take decisions regarding which is first and which is second.

3.5.2 Implementation notices

Furthermore a huge speedbump at the case of Burmester Desmedt is that the definition of it does not display the participant index to start from 0 but from 1 that confuses further the developer who tries to implement it. At the other hand on case of Tree-KEM and on ART there is existing code so a developer with strong analytical skills should be able to use it easier with few modifications such as unit tests or some minor improvements on the tree iteration and storage. At the worst case or re-implementation (for legal reasons) it should relatively easy to perform it in a couple of sprints.

On the other hand both Tree-KEM and ART are relatively new compared to Burmester Desmedt (presented in mid '90s) and mDP+P key agreement protocols (early in 2000's). Especially on Tree-KEM because is a part of a hatching standard, is rather early to use it in a real production system. Therefore further research is required in order to prove its robustness and worth as protocol.

3.5.3 User Movability

Also, when a participant leaves from the communication channel, in case of Burmester Desmedt the whole process is needed to be redone in order to have a secure channel, in case of mDP+P the protocol mDP+S is needed to executed (in other words the same process with a new uid that does not contain the keys and the ids of new participants). In case of ART the process is simpler by updating a tree and because of existing code the implementation is simpler. The most convenient is the TREE-KEM because of empty nodes, practically being able to convert a tree into a forest of minor trees.

Chapter 4

Existing Bibliography on Privacy Enhanced Instant Messaging.

4.1 Introduction

OTR and Signal are the most known protocols when it comes on privacy-enhanced chat. From privacy perspective with the use of end-to-end encryption they achieve *unlinkability* to the participants. An overview of the services offering to the privacy and security are:

1. **Protection against corruption:** With the use of signatures and MACs each message is protected against corruption.
2. **Confidentiality** With the use of end-to-end encryption using a common symmetrical key.
3. **Forward Secrecy** The forward secrecy has these dimensions:
   - **Per-message forward secrecy:** Where on each message a new key is generated, for example SCIMP ratcheting [23] or Signal's Double Rachet [24].
   - **Per Session forward secrecy:** If 2 participants (n-participant in multiparty chat) have chatted before and make a new chat session, if an adversary has compromised older chat sessions cannot access the new ones.
4. **Deniability:** Because anyone can modify a message both participants can deny that has authored a message in a chat session. Also the use of ephemeral keys on key agreements such as Tripple Diffie Hellman or Extended Tripple Diffie Hellman, protocol enhances the deniability as well.

But the challenge is when it comes to multi-party instant messaging there are some extra concerns. The extra concerns that a group chat session has are [15, 25]:

1. **Entity Authentication:** Ensuring that all the participants in the chatroom have the same keys and each participant "sees" that a chatroom is consisted with the same participant.
2. **Origin Authentication:** Ensuring that a user actually has actually authored a message. In other words avoiding the impersonation attacks.

4.2 OTR-Based Protocols

Generic Overview

The direct successor of the OTR protocol is the mPOTR one [25], a later implementation of it with some improvements it is the (N+1)sec one [15, 26]. The mPOTR provides the framework of a privacy- enhanced multiparty chat session. According to mPOTR [25] a multiparty chat session has the following phases:

- **Setup Phase:** At this phase the common secret key is generated and each participants are being authenticated. On user movability (when a users joins or leaves) this process is re-executed. Also a session identifier *sid* is generated
- **Communication Phase:** This is the section the participants exchange the messages on an encrypted channel using the key generated from the previous step. Each message is encrypted and signed before being broadcasted to the common channel.
- **Shutdown Phase** When a participant wishes to leave then some checks are performed on the chatroom. The first one is deliver any undelivered messages, the second one is whether the room is in *concesus* meaning that all participants have received the same transcript.

Setup Phase

Usually on this step a group key agreement protocol will be used in case of (N+1)sec [15] the mDB+P protocol is used as seen in 3.4.3. Whilst for the mPOTR is proposed a key agreement Based upon Bohli's key agreement protocol [17, 18]. A noticeable feature is that on (N+1)sec when a user parts the key agreement for making the new key is executed in parallel and the session continues with the old key for a while, when a new key is agreed then the old one is being ditched for the new one [15]. That leaves a small vulnerable window for message leakage, on the other hand a good user experience is offered due to lack of disrupted communication.

Communication Phase

The end to end encryption is the stronghold of the user's privacy, it ensures privacy *unlinkability* services and are being enhanced with techniques for forward secrecy. On mPOTR is mentioned that a user should encrypt the message with the common key generated in the setup phase and signed with an ephemeral signature key [25]. Also the forward secrecy is achieved in (N+1)sec with the use of temporary keys per chat session [15] also over long sessions the key is updated periodically ensuring in-session forward secrecy if a user is idle (no chat message has been exchanged) then it sends a "heartbeat" message [26].

Shutdown Phase

During the shutdown phase is necessary to tell whether the chat has reached the *concesus* on mPOTR is proposed to sort and then hash all the messages [25], similar approach is used in (N+1)sec the message sorting is achieved via a sequence number [26].

4.3 Protocols that do not use Group Key agreement

Even though OTR-based protocols offer maximum privacy on top of communications there are other attempts that offer privacy enhanced encryption even though use different approach and some of them do not offer all the features of the OTR-based ones.

4.3.1 Anonymous chat - Metadata resistant chat.

An another approach is to offer privacy is via offering anonymity during the chat session. One way to do that is via encrypting, grouping and shuffling the messages like dissent does [27] with the *shuffled send* primitive and tries to hide the author of a message. For encryption and decryption public-private key pairs used that usually are somewhat slower.

An another approach is to use anonymous networks such as TOR and on top of them perform he communication like CWTCH does [28]. it extends the ricochet protocol [29] with the use of untrusted relay servers on top of TOR network. A nice UX feat is the use of multiple devices and consideration of a multi-device use of a single user.

This approach has the benefit that the use of a already-anonymous channel that TOR offers. Also the semi-serverless or the full serverless approach leaves minimum trace on who's contacted with whom, thus the deniability is secured by default. Also the anonymous chat has by default encrypted and integrity protected data exchange so nbo further measures are required. On the downside the TOR ids and urls are not human-friendly as I explained in a previous essay [1].

4.3.2 Naive simple approaches.

Beyond the anonymous and OTR-based approaches there are some approaches using only end-to-end encryption. These approaches do not offer either anonymity or the features of the mPOTR and (N+1)sec. In this section the approaches explained are either performance inefficient or very prone to adversaries.

Signal- WhatsApp

The WhatsApp implementation of the Signal protocol use the so-called *sender keys* [30, 31]. What it actually is done via p2p signal channels each participant sends his/her encryption and integrity protection keys, then they use the key delivered via these p2p channels to encrypt and decrypt the messages exchanged into the room [30, 31]. This approach relies to the capability of the server to handle the message overhead [30, 31] and this approach used in order to avoid the latency occurred by a group key agreement protocol [31].

So for the multiparty communication phase for a n-sized group key there are n session keys that each user derives it own session key utilizing an HMAC-256 algorithm as Key Derivation Function [30] on a KDF chain, as seen in figure 4.1. Though this approach even if derives keys does not offer any forward secrecy [31] compared to the Double Rachet [24] algorithm that offers per-message forward secrecy.

Threema

In the Threema Instant messaging iOS and Android app, the group messaging is handled by the clients by individually encrypted and sending the massages to each one of the user, the servers do not participate in group establishment [32].

This approach is used because on the specs the typical message is about 100-300 bytes [32] and the servers have enough resources to handle the load. Also the use of Salsa Stream Cipher provides minimum latency and media such images, videos etc etc are encrypted and stored into media servers where are being fetched separately [32].

The forward secrecy is offered via generation of new temporary keys when the application starts, also there is no permanent storage for them.

TokTok - Tox

An another protocol is the TokTok one, a p2p IM and VOIP protocol [33]. The group message exchange on top of it is done via establishing an overlay DHT network between the participants [34]. The messages are relayed via pairwise encrypted connection using elliptic curve Diffie Hellman [34].

It uses XSalsa20 Stream Cipher [34] like Threema does [32]. But on the other hand it is prone to malicious nodes that due to pairwise encrypted channels the message is decrypted and encrypted again so a malicious or a compromised node can easily tinker and analyze the messages like a centralized no-privacy enhanced approach.

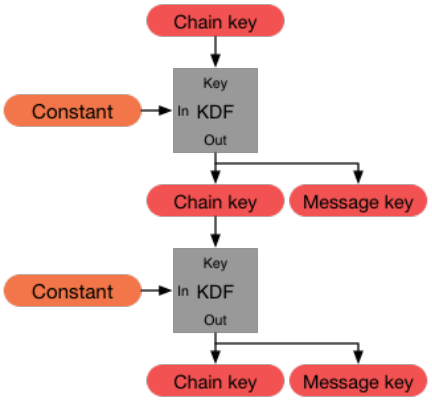


Figure 4.1: Signal's KDF chain used for key ratcheting [24]

4.3.3 Latest Rising Standard Protocol from IETF.

The latest attempts in order to standardize a secure - privacy enhanced Instant Messaging protocol is the MLS [21, 35], a protocol that is currently worked and being developed from IETF.

Architecture

According to the latest draft, the architecture is relied upon the following services [35]:

- **Authentication Service (AS)** A service responsible for storing, indexing and providing each participant's public keys.
- **Delivery Service (DS)** The service where the key agreement is performed and messages are delivered between participants in a group.

Both AS and DS may offered from the same or different providers [35]. The group itself may contain 2 or more participants.

Protocol Overview

The protocol allows these operations to be performed [21]:

- **Participant adding**
- **Participant removal**
- **Participant Key Update** For forward secrecy users frequently update their keys.

For the common key agreement the Tree Kem algorithm is used as seen in the section 3.4.4. The users are sequentially added for a user that initiates the session, during this session the key is being gradually generated and is updated when a user joins to the session.

Also at the spec-in-develop also is taken consideration the post-compromise and forward secrecy security that relies upon the message order that is enforced either from the server or the client. Using that with the combination of a hash ratchet forward secrecy is provided, there's prediction for each message via ratcheting to create a message key [35].

4.4 Overall Comparison between the protocols

So far the only attempt to seriously offer a whole privacy package is the (N+1)sec even though practically has not a huge deployment as a technology. MLS sound promising but it is still on his first steps in order to be formed into a final standard and considered as a production-ready solution.

A huge gap is the lack of having per-message forward secrecy similarly to Signal's Double Rachet algorithm [24] on multiparty chat, the only way that forward secrecy is preserved is per session. The only promise we can have it the TREE-Kem, but as said before is still has a long way to go.

Also on the anonymous chat side, the underlying TOR protocol does not allow to have multiparty meeting points and no prediction for multicast-like communication. Further research and development on TOR for high performance multiparty communication will aid significantly the performance of the CWTCH [28] protocol as well.

On the other side on the naive approaches even though lack some of the (N+1)sec and mPOTR features have been chosen because:

- **Implementation of key agreements is hard.** As covered in section 5.
- **Key agreements offer low UX and are slow.** On (N+1)sec [15] and on mPOTR [25] on user parting a new key agreement needs to re-performed adding communication overhead and latency. In order to avoid that the key agreements needs to be done in parallel offering a small vulnerable window. The most performant one (ART,TREE Kem) are relatively new and need to be tested on production.
- **Messages are delivered out of order:** Due to heterogeneous network speed, and various other factors on a multiparty Instant Messaging the messages are being delivered in different order, that provides a speedbump on the implementation of the algorithms that require syncronicity for example Burmester Desmedt [12].

Chapter 5

BIG BITE: Experience and Lessons Learned upon implementing a group key agreement protocol.

5.1 Introduction

Having read the bibliography I have attempted to implement an existing group key agreement utilizing electron and javascript. The reason why I did this was because javascript nowdays is a industry standard having heaps of libraries and an enormous ecosystem (libraries, frameworks etc etc) and you are able to develop applications relating web, desktop, mobile apps and server ones. For communication layer I used XMPP and XEP-0045 xmpp extension [36] for communication layer.

But due to limitations of the javascript's ecosystem, there was a second attempt utilizing the c language and using OpenSSL library and MPI as the communication layer. On the first attempt the mDP key agreement [14] has been implemented whilst on second one the Burmester Desmedt key agreement has been developed [12].

5.2 Attempt 1: Developing the mDP [14] key agreement algorithm using electron and javascipt.

5.2.1 Library used for XMPP communication and XMPP Problems

In this approach uses the `@xmpp/xmpp` library [1]. This library has been used because the audit for the npm showed 0 known vulnerabilities at the moment where the application has been developed. On the downside it offers only basic XMPP connection features that I had to implement the XEP-0045 support by myself, and that way my first difficulty. In order to transmit the results from the group key agreement the values were encoded as hexadecimal strings (strings containing characters 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F), for encoding and decoding it was used the javascript Buffer API [2]

[1] Npm package: https://www.npmjs.com/package/@xmpp/client
[2] Documentation in https://nodejs.org/api/buffer.html

5.2.2 Problems on implementing the algorithm described from paper into a piece of code.

The first issue was the algorithm selection itself the key size is too big in order to use standard types and operations usually used in variables in order to perform mathematical and binary operations. For example in mDB+P algorithm [14] the Binary XOR need to be performed between at least 1024bit data, and in order to do that I needed to implement my own method. Alternative approach for this problem was the use of `node-bignum` [3] but there were linking issues with the underlying OpenSSL library that is based upon, thus ti was less time consuming to make a method to just do that.

5.2.3 Problems in key generation

The default Node.js API (electron is built upon chromimum browser and Node.js) used for Diffie Hellman operations when not provided the parameters p and g takes some time to generate key values. On the other hand once the values transited and used the key generation becomes much much faster. That resulted a freeze on the developed desktop application providing poor User Experience.

In order to overcome it I used the `threads` [4] library in order to execute the key generation in a separate thread, but it came with the cost that I had to re-create the Diffie Hellman object after the thread execution.

5.2.4 Problems in Key agreement itself.

The first problem I needed to avoid is a key non-agreement due to out of order join messages thus I sorted the friends by its nick in a lexical order. Even though I did that I came across with another problem that is causes by a failure on checking whether $z_1 \bigoplus z_2 \bigoplus \dots z_n = 0$ even with the absence of any corruption. Also a big hurdle is to calculate the z' values used in the final key agreement step, that is because there's a hard understanding on the translation between U_i annotation and the approach used in the actual implementation (the users have the form `user@example.com`.) Even in the best approach used there was no common key generated each user had a separate key.

5.3 Attempt 2: Use MPI, C to perform a Burmester Desmedt

Therefore in order to avoid any trouble due to time limitations, I decided to use plain C and perform the original-intended Durmester Desmedt algorithm. In the previous attempt I choose the mDB+P [14] instead of Burmester Desmedt because the lack of easy-to-use APIs in electron environment, also the mDB+P algorithm [14] allowed me to use the standard API offered by node.js (can be used in electron as well). Furthermore OPENSSL library has a API for arbitrary-sized integer math and they delivered by default on any GNU/Linux distribution.

5.3.1 OpenSSL Limitations

In my system I had an earlier OpenSSL library especially the `1.0.2g` thus I needed to manually build it taking care to keep it in local project scope. So I added the OpenSSL as a git submodule to my project [5]:

```
git submodule add --depth=1 https://github.com/openssl/openssl.git
```

Afterwards I manually build the library using the following commands:

```
cd ./openssl
./config --prefix=$(pwd)/../builds/openssl --openssldir=$(pwd)/../builds/openssl
make && make test && make install
```

[3] Seen in https://github.com/justmoon/node-bignum
[4] Npm package: https://www.npmjs.com/package/threads
[5] Project source code located in https://github.com/pc-magas/burmester-desmedt

As you can see I used the `pwd` command in order to create full path that the config script required for the library installation. And I installed it into a project specific folder that has been already ignored from the version control system (git), afterwards I compiled based upon a stackoverflow answer [6] using a Makefile. But this approach proved to be time-consuming thus I decided to use a virtual machine having a later version of OpenSSL, the virtual machine used the latest LTS version of Ubuntu 18.04.1 that is delivered by the OPENSSL version 1.1.0.

5.3.2 Problems Understanding the final step

As the attempt above the hardest part is to calculate the following equation from Burmester Desmedt's algorithm $K_i = (z_{i-1}^{nr_i})X_i^{n-1}X_{i+1}^{n-1}\dots X_{i-2}$. In order to solve that problem code-wise I did the following:

- I calculated the $z_{i-1}^{nr_i}$ where n is the group size.
- I stored the X_j values where $0 \leq j \leq n$ into an array.
- I cyclic iterated the array calculating $K_i = K_i * X_j^s$ between $i, i-2$ and $s = size - 1$. Afterwards I reduced the s by 1.

And in the end I failed to make all the parties to have a common key.

5.4 Lessons Learned

A generic advice I can offer from this experience is to implement group key agreement algorithms in, C++ (that can be linked with C libraries as well) and use the plethora of C and C++ libraries offered allowing you to perform mathematical operation with Big Integers. Afterwards with the use of Bindings put it altogether with your Javascript application, this approach allows you to control the linking with the libraries and to manage the final application building without the need for workarounds and issues with 3rd party libraries that the developer does not control.

Also for these libraries, for example the OpenSSL, lots of help is offered through online communities such as stackoverflow [7] allowing to has a development with less troubles. Furthermore, in case of C/C++ builds on application utilizing the OpenSSl library on GNU/Linux, we should be wary on system's the OpenSSL version. Because older but secure versions of it may cause incompatibilities thus depending the case we should do the following:

- In case of a docker-contained of a flatpack, snap [8] packaged application, we may need to deliver and a pre-built compatible OpenSSL as well.
- Also in case of a traditional packaged application (eg in a .deb package) we should make the code to the wary of the OpenSSL version and provide the appropriate way to access the their methods depending the OpenSSL version.

Furthermore in case of using bindings in Javascript and Electron the same principle applies as well. In case of using of an Object Oriented language the use of Design Patterns such as the implementation and proxy [37] one comes very handy in this case, especially when making an API for a Javascript application using C++ bindings. Furthermore considering using newer group key agreements that have been already implemented in code are useful as well such as the ART and the Tree KEM.

Also is advised to use code that already implements a key agreement, a group key agreement requires lots of effort because even if on paper and mathematically represented seems fine at the end of the day it has to be implemented on code in order to have some functional worth.

[6] https://stackoverflow.com/a/54548607/4706711
[7] https://stackoverflow.com
[8] The flatpack and snap are attempts to have distribution-agnostic self-contained applications in GNU/Linux

Chapter 6

Initial Architecture of Himitsu project.

6.1 Introduction

On my previous essay I presented an idea that allows users to advertise to their friends services [1] called *Himitsu*, with its name is derived from the Japanese word standing for secret. Specifically the Himitsu project has been defined as [1]:

> *Himitsu project is a set of existing and new standard protocols, api's and applications aiming to offer UX aware privacy.* (Dimitrios Desyllas)

From a more technical aspect it will extend the privacy enhanced chatting into a reverse connection end-to-end encrypted real-time data exchange, allowing multiparty real time data exchange offering a smooth experience between user's devices.

6.2 Subsustems of Himitsu project

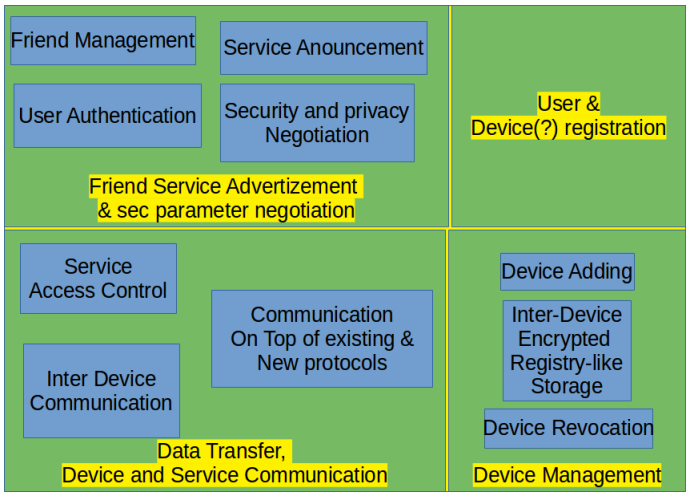


Figure 6.1: Subsystems of the Himitsu project

Based upon the studied bibliography mentioned above, I decided that the Himitsu project will be consisted from the following subsystems as seen in figure 6.1:

- **User registration and authentication** Where a user will be registered and authenticated.
- **Friend Service Advertisement and security parameter negotiation.** This subsystem will be responsible to offer an end-to-end publish subscription announcement on top of XMPP. When a user wishes to connect into a published service a new security channel will be negotiated on top of an existing secure one.
- **Device Management** With this subsystem a user will be able to use an existing authenticated device to add and remove his devices. Also using XMPP a device will be able to communicate with other devices as well for inter-device services eg. the smartphone to be able to communicate with fridge.
- **Data Transfer** The collection of existing or new protocols used in communication between users and devices. On this layer will be application-specific protocols and way to provide per-user and per-device access control combined with the *Friend Service Advertisement and security parameter negotiation* subsystem. Here a major role will play a new in-house developed protocol named *proxy-pass* enabling the users to have end-to-end encrypted and privacy-enhanced real time communication using reverse connections.

6.2.1 Friend Service Advertisement and security parameter negotiation.

In this subsystem a user will be able to connects into his/her friends on a pre-secured channel. This subsystem will offer the following functionalities:

- Creation of a **friendship master key** during friend adding. When a user adds a new friend this will be used as master key that will be derived on per-service.
- Creation of a **group master key** in case of a group creation. This key can be either an ephemeral one or a permanent one depending each application's needs.
- **Access control**, when a user desires to connect into a service then a parameter negotiation will be used, in this section the owner of a service will allow or block a device or a user to connect into a specific service. A successful connection will derive an ephemeral connection symmetric key.
- **Service Advertisement** A user will be able to advertise its services to its friends or groups that has allowed to be advertised. The advertisement will be an encrypted message directly sent to groups or friends.

Services

The services is a flexible way of telling any way that 1-n friends or can interact with 2-n devices. Some examples of what a service can be are:

- A voip and instant messaging application that utilizes a reverse connection.
- The fridge that reminds to his user's smartphone that is out of milk.
- An application used in order to play remotely with you friends Dungeons and Dragons.
- etc etc

Also in this system may be used to perform further security and privacy related actions for example key renewal, ephemeral pseudonym transfer in case that there are services that need some (pseudo)anonymity as well.

The security parameter negotiation will be performed with one of these ways:

- **Direct key transfer**: Using a friend-secured channel will be performed a direct key transfer for authentication and encryption.
- **P2P key agreement**: Used in 2 party communications (without limited to it) using the Extended Triple Diffie Hellman on an already secured channel a new key will be derived.
- **Group key agreements**: On group communications a new key agreement may be used to negotiate a new service key.

In case of direct key transfers first an end-to-end encrypted channel will be establishes via Extended Triple Diffie Hellman and afterwards any key will be exchanged also in case of further security handling the same encrypted channel will be used as well.

Furthermore a service will have be able to advertised from its *service owner*, the user who publishes and offers a service. There are will be 2 types of services:

- **Inter-device services** *Service Owner* owned services that only the service owner-owned devices will be able to communicate with it. For example a supermarket-list service. In case of multiple owners it would be registered to the service outside the Himitsu. The security will be relied upon the derivation of the **Inter-device Key**.
- **Normal Service** A service that will be shares between friends or groups. The security of ti will be relied upon either the **Master Group Key** in case of group or **Master Friendship Key**.

The name of the service in the Himitsu project will be in a n-sized random generated *system name* containing numerical digits from 0-9 and Latin characters from A-Z and a-z. But there are will be *human names* where the user will able to use them in order to access the service. For example a *human name* can be the service title for example "Supermarket list" or the url "http://mysupermaket.onion", a *system name* will be the "A019BBZ". The software running by the end user will be held responsible for converting some defined *human names* into *system name*.

6.2.2 User and Device registration

User Registration

A user will be registered towards Himitsu service either via a web panel or from a client from the device itself. During the first registration will be created at least these 2 asymmetric keys:

- **Device Confidentiality Key** as explained in 6.3.1.
- **Master Signature Key** as explained in 6.3.1 as well.

In case of a web registration the server may either generate the user and master device's keys for maximum user experience. Alternatively for more experienced users the user itself may need to upload the key. Furthermore in case of registering directly from client a temporary code will be sent via email (or SMS) as many desktop and mobile application do nowdays.

User authentication and identification.

The user will be identified by an *user address* that will have the following form `username@server/device_id`. Where `username` is the user's identifier and the `device` indicated the device that a user having `username` is logged in via `device` device. For maximum privacy the `device_id` will be a randomly generated string having maximum 10 letter, containing latin characters from `0-1, a-z, A-Z`. It should not contain any device name, so it incapacitated the ability for the server to figure out whether on smartphone or any other device.

The authentication towards Himitsu of the user will be performed on 2 axis. The first axis the device is authenticated and on the second axis the user itself is authenticated. A challenge response scheme will be used for authenticating

the user. Based on the `username` and `device_id` the server will fetch the correct **Device Confidentiality Key**. Each challenge from the user will be encrypted with device's **Device Confidentiality Key** and signed with **Master Signature Key** thus both user and device will be authenticated as seen in figure 6.2.

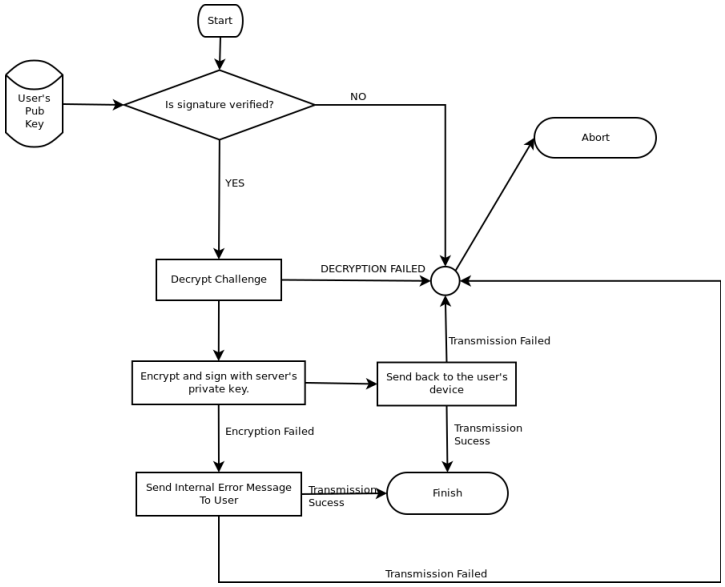


Figure 6.2: Flowchart explaining how server authenticates the user's challenge

Furthermore in case of a web panel conventional username password based authentication will be provided, and the user will be authenticated ONLY to the web panel using that. Each client will be authenticated via client's keys. An alternative approach will be a temporary authentication link sent via email only in case of a account recovery due to device loss.

Storage of user's personal info.

The minimum required personal information for user authentication will be email or a phone number. Also there will be a *persona* service for the user to manage his/her display information each persona will be available to a subset of user's using a specific service. So depending the case then no unwanted information will be available.

In case of *persona* the server will be trusted to serve the correct one, thus it may be a privacy leak in this case but it will be the minimum even in worst case.In order to mitigate the risk optionally an advanced user may need to host a dedicated *persona server* in order to have maximum access control towards his privacy. Furthermore due to decentralized nature of the **Friend Service Advertisement and security parameter negotiation** will be able for the user to move his data to another one even into a self-hosted one (in case of advanced users).

Account recovery during a device loss.

There are will be 3 ways for the user to recover its account.

- Via using an another *registered device* and deleting the compromised or lost one.
- Via the user of a web panel either via authenticating via username and password
- Via a web panel with a temporary authentication link and one time password (also known as OTP) using his email or phone number.

The most preferable approaches would be either the use of a web panel utilizing the One time Password (OTP) or the use of an another registered device, because it offers a good balance between user-friendliness and security without the use of passwords.

In case of using OTP the following steps will be followed:

- The user requests account recovery.
- Web Panel sends a link where the user will be authenticated. This link will expire after a time interval.
- User visits the link.
- Web Panel sends a one time password.
- User types the one time password to the visited link and logs ins to the web panel.

Also the security will be relied upon the single use of the one time password and the short interval for the user to login. Also the only actions that will be available on is the access revocation of a registered device. Further security armoring would be the use of captcha first on the request for password recovery and secondly after a failed attempt on entering the OTP. Also after a fixed number of failed attempts the process will need to be re-done and the captha will be present on each step of account recovery.

6.2.3 Data Transfer

On data transfer layer will be used any available TCP/IP Application Layer protocol, it can be either a typical client-server connection as seen in figure 6.3a, or a TOR hidden service or even a reverse connection using our own made Proxy Pass protocol as seen in figure 6.3b. Furthermore the Proxy Pass protocol can be used also for same-user owned devices as well. Furthermore the special treat is the multiparty reverse connection capability as seen in figure 6.3d.

Proxy Pass protocol

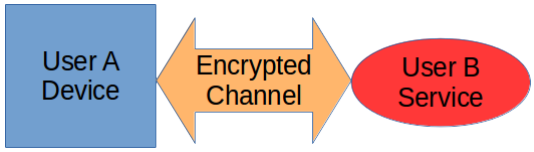
The proxy pass protocol as mentioned above is a special protocol used for end-to-end data transfer utilizing reverse connections, it can be used either as proxy data transfer or as a TCP/IP reverse connection abstraction layer enabling data transfers like a typical Server-client connections even on environments where there it not possible to publish a network port (eg. intranet, NAT).

All the traffic will be exchanged both via an encrypted channel to the server and end-to-end encrypted via an encrypted channel to the end user as well. Also is considered both message-driver and stream-driven data flows as well. Each user can assign a *room* a mnemonic name for letting the server know in who user to deliver the data, furthermore a room's data exchange can be separated in *streams*. Each user wishing to connect into a room can be authenticated via a secret pre-defined ephemeral credentials or tickets, optionally a user can be authenticated to the server as well.

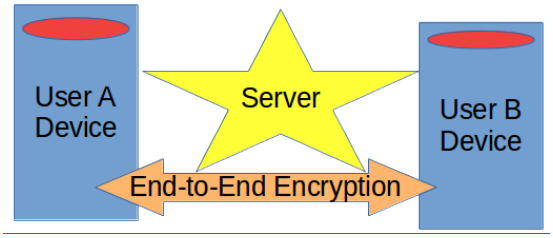
By using that protocol a huge variety of services can be offered with awesome user experience for example:

- VPN services for example in a corporate intranet or for "entertainment" usage.
- Peer to peer gaming, instant messaging and voip and file exchange without the need for specialized server (eg. playing counter strike without the need for server and Hamache [1]).
- Peer to peer DNS without the need for a domain name registrar, useful for corporate intranet- only DNS (in case of VPN) and for DNS spoofing bypass (used mostly by ISPs for "taking down" sited for legal reasons).
- Smart home applications such as turning on the heater or the air condition.

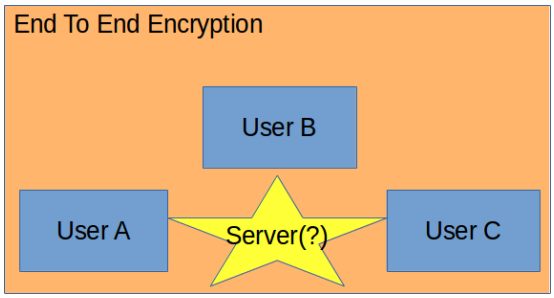
[1] Available at: https://www.vpn.net/



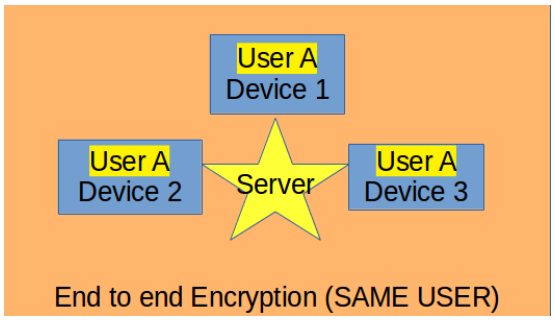
(a) A service using a dedicated server for it.



(b) Peer 2 peer communication using proxypass protocol.



(c) Multiparty Proxy Pass communication.



(d) Same owner owned multiple devices communicating via Proxy Pass protocol.

Figure 6.3: Himitsu Data Transfer methods

6.2.4 Device Management

Device registration and device chain

Each user will contain a *master device*, the device used to register at the first time to the Himitsu system. Each device registered to Himitsu it will be noted as a *registered device*. A *registered device* can be used to register other devices the user owns via a QR scan, bluetooth, voice encoded data transfer or LAN (including on-the-fly wi-fi generation). By doing that a key transferring will be performed on top of a TLS channel will be performed. In case of audio data transfer where appropriate volume short distance and short time will be used for data exchange. The *master device* is also a *registered device* as well.

Alternatively in case of the 2 devices being on the same LAN then it can be used instead of a QR scan as well, in this case depending the scenario a network port will be opened instead for a short amount of time, one the key transfer is complete then the port will be closed as well. There are 2 scenarios when a user adds a new device to Himitsu either bv scanning the QR code of an unregistered device to a registered one or the opposite, a registered device scans the QR code of an unregistered one.

In the first one an unregistered device scans a QR code of an registered one, for example a user wants to be able to connect into the Himitsu project via his smartphone and has been registered via desktop or laptop, then the following steps will be followed as explained in figure 6.5:

- The unregistered device either connects to the registered one via LAN or scans the QR code.
- The Unregistered device transfers his asymmetric public key(s) to the registered device.
- The registered key sends back any other symmetric and asymmetric keys.
- The registered device publishes the unregistered devices's public keys to the server.

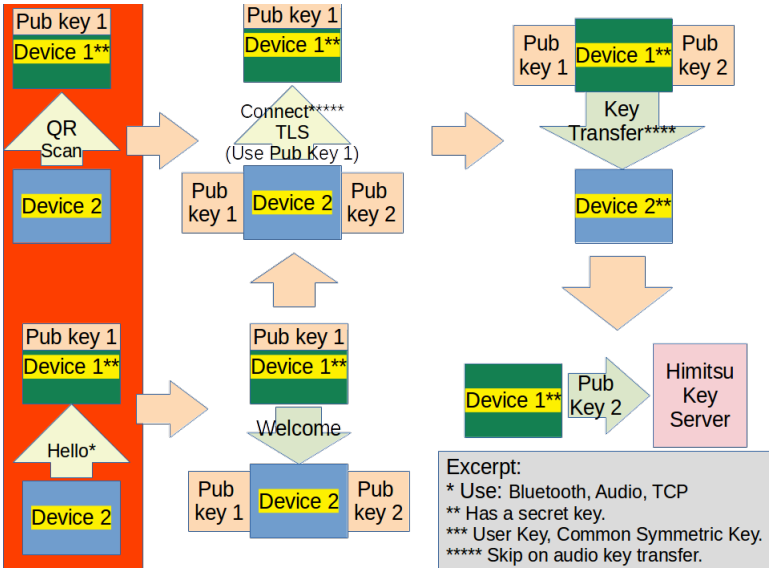


Figure 6.4: A user scans an unregistered device from a registered one.

In case of the opposite, meaning a *registered device* scans the QR code of an unregistered device, for example a used has been registered via smartphone and wants to connect back using his desktop. Then the device's public key will be transfered either via LAN first or via QR code after the TLS session been established then the devise will only transfer any key that has stored to the new device and then it will register it back to the Himitsu.

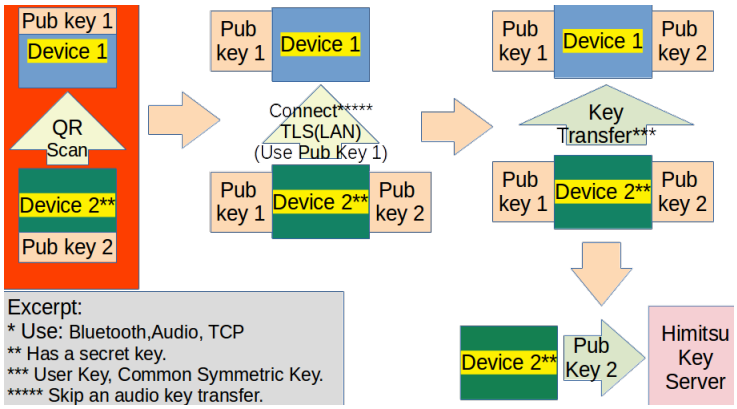


Figure 6.5: A user scans an unregistered device from a registered one.

As you can see the security of this process is relied upon these factors:

- The key transfer session will be a short one.
- The data transfer channel will be secured beforehand via a TLS connection.

For device deletion will be performed from a *registered device*. Also the *master device* can be deleted from a *registered device* as well, without any much impact on user's user experience.

Common Device Encrypted Cache:

For maximum user experience and smooth transition between devices a key-pair json-like storage is introduced. All the data will be encrypted with a master symmetric *device key* and will be double signed with a *Device Signature Key* and the *Master Signature Key*. In case of device public keys then no encryption will be applied at all.

A major flaw of using this approach is due to device loss and compromise, a potential malicious user may have access to this cache. So in case of device deletion the access to data should be denied. Also the common cache is needed to be designed in order to have Backwards secrecy so in case of key loss and key compromise the data should be non-accessible to the adversary.

6.3 Keys and Key Management

The key management for the whole project is the one where the whole security and privacy will be relied upon. Both asymmetric and symmetric keys will be used in order to secure any communication. Also for maximum security any symmetric key will be derived ans it will have specific purpose. Also is planned an extensive use of ephemeral keys as well (for example in key agreements).

6.3.1 Asymmetric Key Management

A user will contain at least these sets of asymmetric key pairs:

- **Device Confidentiality Keys**: Keys where will be used when a user will need to connect with his device to the Himitsu. Each device will have his own keys.
- **Device Signature Keys**: In cases such as inter-device cache in order to detect any possible corruption of data these keys will be used to protect the data integrity.
- **Master Signature Key**: Keys that will protect the integrity and non-repudability duting Key agreements.
- **Asymmetric Ephemeral Keys**: Keys that can be used where deniability needs to take action in case of Triple Diffie Hellman key agreements.

Each *Device Confidentiality Key* will be signed using the *Master Signature Key* ensuring that the user is the owner of this key. In other words, the user will generate self-signed certificates for his devices. In case of device adding the registered device will sign the user's key before publishing it into the Himitsu server.

6.3.2 Symmetric key management

Also the user will have the following symmetric keys as well as seen in figure 6.6:

- **Master Friendship Key**: That will be a product of a Triple Diffie Hellman on top of a end-to-end TLS channel.
- **Master Group Key**: That will be result of a group key agreement key.
- **Master Service Keys**: Keys derived either from *Master Friendship Key* or *Master Group Key*. Each service will contain one key.
- **Session Keys**: Keys that will be generated per session when a user connects into a service, a session key will be an ephemeral one derived either via a *Master Service Key*.
- **Inter-device Key** A common encryption key for all devices.

Also in case of proxy pass protocol a *Session Key* can be derived to a *room key* afterwards also it can be derived into a *stream key* where with further derivation will protect the data exchange confidentiality and integrity. Alternatively it can be derived into *Session Confidentiality Key* and into *Session Integrity Key* as seen in figure 6.6. Furthermore the service key regardless the data transfer approach will also be derived into a *advertisement key* where it will be used in order for an *service owner* to advertise the service. In these keys no forward secrecy will be offered but this key will be ratcheted via a KDF chain for a good balance between security and user experience.

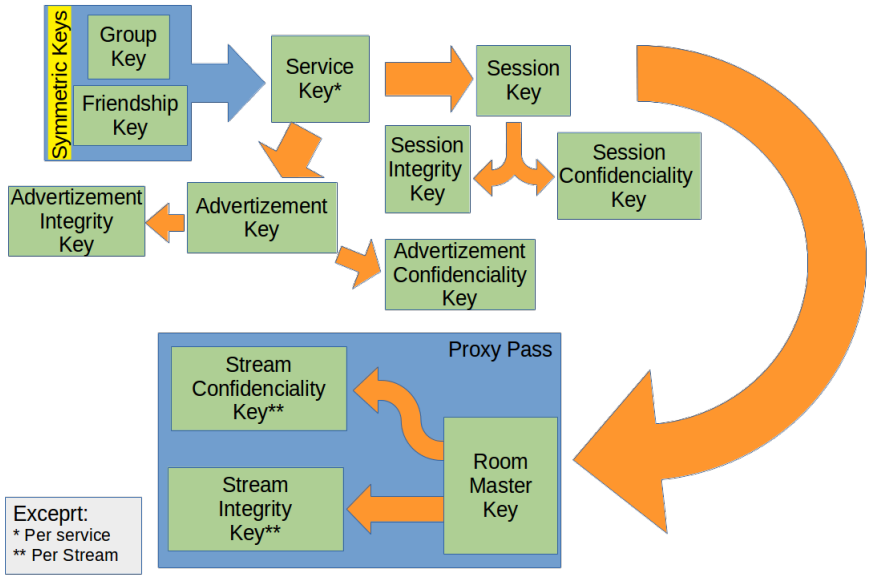


Figure 6.6: A symmetric key derivation map.

6.4 Software Architecture and protocols used.

6.4.1 Uses of XMPP protocol

For both the *Friend Service Advertisement and security parameter negotiation* subsystem and the *Device Management* subsystem, as mentioned above, the XMPP [38] protocol will be used. The reason why this protocol will be used according to the battle scars earned from the chapter 5 is because XMPP [38] by spec allows you to create custom protocol extensions, useful in case of key agreements.

Also there are various official protocols extensions allowing us to offer tailor made data exchange. Furthermore XMPP protocol is used either using TCP and http using either via the utilization of websockets as per RFC7395 [39] or using the so-called BOSH as per XEP-0206 [40]. Also the variety of existing officially recognized protocol extensions will make the development easier, the list of the useful for the project official XMPP extensions are listed in table 6.1.

6.4.2 Applications Supporting the projects.

Agent

Each device will run a specialized NT service like/ Unix daemon program (defined onwards as *agent*) that will offer a connection layer and abstraction between the rest of HIMITSU network services and the application. Therefore application developer will just need to implement the intermediate layer provided from the service.

XMPP Extension	Usage
XEP-0045	Possibly used to manage groups and perform group key agreement.
XEP-0049, XEP-0223	For key, and profile information storage.
XEP-0222	For public Key storage
XEP-0249	For service and group invitations.
XEP-0055	For friend and public key search
XEP-0060	For service publication

Table 6.1: XMPP extension usage in Himitsu project

The communication between application and agent will be performed via sockets. The protocol used on these sockets depends on the application type and the usage scenario. The sockets will be separated on these types depending their use:

- **Management Socket**: This socket will be used by default in order to fetch add and remove friends and groups, add and remove devices.
- **Proxy Pass Socket**: A dedicated socket used to transfer data via Proxy Pass protocol.
- **Proxy Sockets**: Dedicated sockets offering traditional proxy services allowing to pass network traffic either via Proxy Pass protocol into external endpoints. So cases such as region-restricted services via ip will be lifted.
- **Application specific sockets**: Sometimes depending the usage scenario it may need to simulate applications such as http, smtp etc etc. In order to do that the device will provide the necessary sockets to the standard ports in order to do that offering awesome seamless user experience.

These sockets will be either local-only on user's device (listening into loopback interface) or lan-only in case corporate usage.

At least the *Management Sockets* and *Proxy Pass Socket* will be developed as open source and any api and protocol will used on it will openly documented and provided under a GPL-compatible, OSI approved, Free and Open Source license. Furthermore, an api for modules and plugins loaded from the agent in order to offer *Application Specific* sockets will be offered as well. The modularity is a key feature both from technical but from business perspective as well.

The usage scenarios covered for the agent are the following:

- **Service First - Proxy Onwards**: Where in order to connect into a service first its security first and afterwards using either the *Proxy Pass Socket* or *Proxy Sockets* will be connected to the service as seen in figure 6.7b.
- **Service First - Direct Onwards**: Where the security parameters will be configured first and onwards a direct connection without any need to connect through Proxy Pass protocol as shown in figure 6.7a.

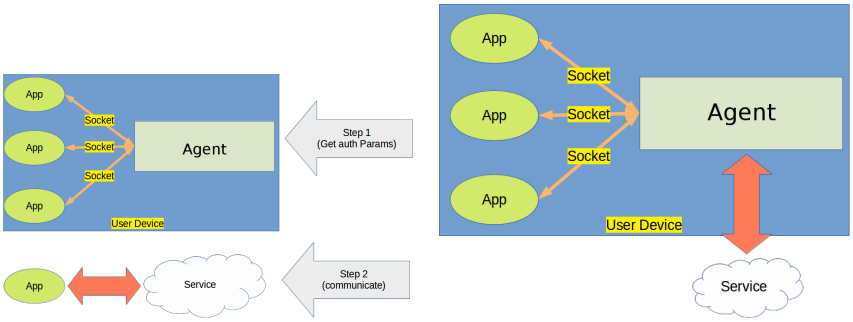
Any application that needs to be able to interface the agent's sockets therefore provided the correct APIs each application distributor will be able to extend the user experience offered by Himitsu.

Furthermore the agent will be responsible to convert some specific *human names* into *system names*.

Himitsu Core Server

This server will be offer the following services to the *agent*:

- **User and Device registration**
- **Friend Service Advertisement and security parameter negotiation**
- **Device Management**



(a) Via Direct connection (b) Via Proxy Pass protocol

Figure 6.7: Scenarios for connecting into a service

For the **Friend Service Advertisement and security parameter negotiation** will be able to contact with other *Himitsu Core Servers* allowing for the user to select its own provider that trusts. Also it offers a better user experience allowing for each user to connect with any other user without the need for it to have more than 1 accounts. More information regarding technologies is covered into 6.5.2

6.4.3 Himitsu Settings and Management Client

For interfacing functionalities such as friend and device management, an electron application (from onwards will be defined as *frontend*) will be developed in order to offer a user interface for them combined with the agent mentioned above. This option allow for having same functionalities with different look and feel and maximum modularity, combined with the openness of the APIS, that will allow for everyone to develop their own fronted app.

A major user experience goal is to offer the ability to extend and migrate existing user connections from existing social media, web applications (eg. github), and email and phone contacts in case of mobile app, into Himitsu. Thus a "social extension feature will be offered extending the capabilities of existing known web applications. But this may reduce just a bit their privacy as well so by allowing 3rd parties to develop their own fronted we allow this risk to be mitigated, also in order to lessen the privacy loss optionally TOR will be utilized as well.

6.5 Libraries, programming languages and cryptographic primitives.

6.5.1 Libraries for cryptographic operations and cryptographic primitives.

For key agreements the most suitable candidates are the (extended)Triple Diffie Hellman [9, 15] for single party key agreements or as a part for multi-party key agreement, such as the other candidate for it ART [10]. As far regarding group key agreements the Tree Kem [11] will be used as well.

For the proxy pass stream-based data exchange the XSalsa20 will be used. In case of a proxy-pass message-driven data exchange it can be used the AES in Galois Counter mode with the use of a Double Rachet-like key for per message forward secrecy.

Hash functions are candidates for key derivation, alternatively Key derivations functions and modular exponentiation can be used as well. For any cryptographic actions there will be used either the Libsodium [2] or the OpenSSL.

[2]Documentations and information https://download.libsodium.org/doc/

6.5.2 Miscellanous Libraries and technologies

For the agent a possible candidate is the use of node.js with C++ bindings for cryptographic operations and multithreading in case of cpu-consuming operations. For the frontend will be used electron with typescript and for the frontend's visual and graphical user interface the React with Bootstrap will be used.

For *Himitsu core server* XMPP servers will be used offering some flexibility allowing openfire, ejaberred and prosody xmpp servers to be used with the use of specialized plugins [3] [4] [5] developed by us. Also some a custom xmpp server with the use of nodejs will be used as well.

Also the required XMPP interfacing API for the custom home-brewed *Himitsu core server*, it will be considered to develop our library, because at this time there is not a maintained api for it offering the required level of modularity. For the proxy pass server due to the needs for high performance and low latency in considered the C++ language to be used, furthermore for maximum efficiency the options for a dedicated vm developed upon a unikernel [41] virtual machine is viable as well.

6.6 Minimal privacy and Security Requirements

6.6.1 Privacy Requirements

The Himitsu Project should be able to fulfill some minimum requirements during the end-user's privacy and security. We assume that the worst adversary is each owner, maintainer and administrator of the *Himitsu Core Server*. This type of adversary or similar it should **not** be able to:

- To find out any non Himitsu or proxy pass addresses.
- To see and analyze the contents of a data exchange. Either a proxy pass or a normal communication it should not be able to find out:
  - The private asymmetric or symmetric keys used for encryption.
  - The contents of a communication to a service.

  Even if some communications is compromised the adversary should be able exploit very few messages and after a while it won't be able to read and analyze any extra message.
- To be able to read the contents of the inter-device cache, even on the case of a device loss.

These privacy requirements will be fulfilled using the following mechanisms:

- **Data exchange separation**: By separating logically the data streams we ensure the minimum data loss or compromise. As we can see each data exchange will have its own encryption key also further separation is done by utilizing separate communication channels as well with the use of services.
- **End-to-end encryption**: Any data exchange will be end-to-end encrypted, also the same applies on the key exchanges or key delivery as well either with the use of a symmetric or asymmetric keys.

The most dangerous situation on the user's privacy, considering my current knowledge of the current initial form of the Himitsu Project, is the inter-device storage. It stores crucial data for making the Himitsu a smooth and the whole user experience is relied upon this storage.

[3] For the openfire server plugin api http://download.igniterealtime.org/openfire/docs/latest/documentation/plugin-dev-guide.html
[4] Ejaberred extension support in https://docs.ejabberd.im/developer/extending-ejabberd/modules/
[5] Prosody module api https://prosody.im/doc/developers/moduleapi

6.6.2 Security

The Himitsu's security will be relied upon the lack of passwords, instead the user will be authenticated with the use of keys and challenge-response schemes. The most dangerous situation is towards the security is considered the user account recovery on a complete device loss due to these reasons:

- Even a temporary one use password it still a some sort of authenticator is being transmitted in plaintext form upon a non-secure channel.
- Also not only the authenticator is being transmitted also the connection endpoint is being transmitted as well using the same non-secure channel.

But on the other hand an awesome user experience is provided and the risk of a complete account loss is mitigated. Even on similar systems such as the Signal one [42] there is NO way to recover the account on a device or number loss.

Chapter 7

Results, further research and development.

7.1 Results and lessons learned.

As you can see above, I managed to create an early overlay architecture for the Himitsu project and indicating the technical directions for the implementation of the project. Also I managed to find gaps on existing bibliography that also are technical gaps and difficulties that the Himitsu project offers a great motivation to be solved.

A second result beyond the scope of this essay is that sometimes the seemingly on paper simple actions prove to be the hardest ones, especially in the case of the implementation of group key agreement even if translated as close as possible as described paper's definition still failed to make the participants to agree into key.

7.2 Recommendation for further research in key agreements.

First of all there is the need for an overall comparison on the performance of the protocols in situations with high user movability for example IRC (a huge amount of participants parting and joining at the same time). Also in cases of high user movability it may need to implemented more efficient algorithms for securely changing the key.

Also some research is needed on how the order may affect the protocol's security, for example in case of mDP when 5 users $(U_1, U_2, U_3, U_4, U_5)$ places in a cyclic group like U_2, U_5, U_3, U_5, U_1 how the key agreement is affected without the extra piece of code to sort them out. In the end further research is needed to be performed on the security of the Tree-KEM because it is an early protocol and thus there's no test base to test its strength. The same principle applies to the ART as well.

7.3 Recommendations for further research on Privacy Enhanced IM algorithms.

The privacy enhanced chat either in a 2-party or into a multiparty chat has proved its worth and is essential to the modern day. But in a IM a message is pretty much a n-byte chunk of data that is transmitted all together. But there's no research on how to apply these principles from a message driven-communication to a stream-based communication like VOIP or even in real-time IoT.

Furthermore more research is needed on how to double ratchet keys in the case of a group instant messaging exchange, yet alone when data are streamed instead of being transmitted in fixed chunks in a way allowing the minimum latency during the communication.

7.4 Steps required to be taken for Himitsu project implementation.

First of all there is need for API's and libraries to be developed as well in order to perform key agreement cryptographic operations and that's because having an easy to use API will lessen the burden of the developers and contributors of this project. Of course they will be relied upon existing libraries used for cryptography such as OpenSSL and Libsodium.

Afterwards some research is needed in order on how the double ratchet algorithm should be modified or used in order to fit on a multiparty end-to-end encrypted communication. As a result of this research will be a library for key ratcheting as well delivered alongside with the library mentioned above.

Also some research is required on how the common cache will store the data in order to offer backwards secrecy using a long-lived symmetrical key as well. As mentioned above the common cache is essential for the seamless user experience on privacy that Himitsu thrives to offer.

Further research would be performed on account recovery for passwordless authenticated systems with the use of a system generated ephemeral one time password, a necessity not only on Himitsu but for other projects as well, such as ReCRED [43].

Then there is a need for a standardized XMPP extension for normal and group key agreement to be developed, that's because is a major need for it. If there is a standard way to perform a key agreement then not only the Himitsu protocol but many other protocols will be benefited as well. Also there is an another reason to do so in order to have less number of messages exchanged in order for efficiently a group key agreement to be performed as well.

Finally once these apis reach a significant amount of maturity then in parallel will be developed both the fronted the agent and the proxy pass server. Also further research will be done on the most popular xmpp server solution and a plugin will be developed for it. The initial platforms of choice is GNU/Linux, Windows and android afterwards there will be a move towards apple-based platforms as well.

Appendix A

The Triple Diffie Hellman Algorithm [9, 15]

The Triple Diffie Hellman key agreement, is a way of performing a 2-party cryptographic key agreement, it can be based either on traditional Diffie Hellman or Elliptic Curve Diffie-Hellman. Is is focused on offering improved deniability in a less complex architecture provided from this algorithm [9]:

- 2 users A and B need to exchange a message in a private manner so no one can read and modify their message. Thus each user have their following keys:
  - User A has a Long-term signature Key A and ephemeral encryption key a.
  - User B has a Long-term signature Key B and ephemeral encryption key b.
- Each user signs the ephemeral key with their long them key and then they perform the key agreement.

After that each message is MAC protected using derived from the common secret key for integrity protection. Thus each user can deny that has sent a message.

Furthermore a more advance approach is the Signal's Extended Triple Diffie Hellman [9], in that case both Bob has the following public Keys [9, 44]

- An identity key $PublicIK_B$.
- An signed prekey $PublicSPK_B$ signed with the key above.
- One time keys $PublicOK_B$

At the same time when Alice wishes to generate a common key with Bob then generates an ephemerals key pair $(PrivateEK_A, PublicEK_A)$ and the her identity keys $(PrivateIK_A, PublicIK_A)$. Then the Alice after fetches the Bob's public Keys $Public_IK_B, PublicSPK_B, PublicOK_B$ performs the following Diffie Hellman key computations [9, 44]:

- $DHK_1 = DH(PrivateIK_A, PublicSPK_B)$
- $DHK_2 = DH(PrivateEK_A, PublicIK_B)$
- $DHK_3 = DH(PrivateEK_A, PublicSPK_B)$
- $DHK_4 = DH(PrivateEK_A, PublicSPK_B)$

Afterwards using a Key Derivation Function, Alice, calculates the secret key $S = KDF(DHK_1||DHK_2||DHK_3||DHK_4)$ and sends her public keys to Bob. Bob now generates the same keys as well and is able to communicate.

The (N+1)sec protocol [15, 26] uses a simplified version of this protocol as seen in figure A.1 shows. Both approaches achieve the following [9]:

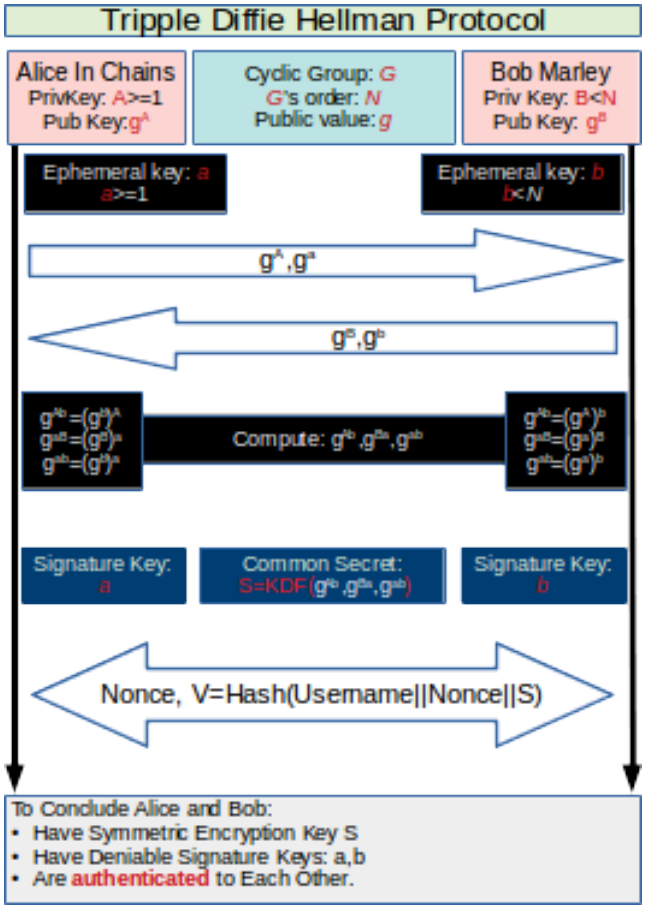


Figure A.1: The Tripple Diffie Hellman Protocol

- **Deniability** With this use we are able to forge an exchanged message sequence. Let suppose a user C wants to forge Alice's messages then he/she/it could retrieve her public key (as seen in figure A.1) and generate an ephemeral key c. So Alice is able to deny that she exchanged a message sequence.
- **Simplified Implementation** Instead of using signature algorithms we use easier to use and faster Diffie Hellman key exchanges.
- **Simpler protocol implementation** With that there's no need for publishing old MAC's (for rejecting replay attacks) thus we can send less and smaller messages.
- **Forward Secrecy** Even a long term key has been compromised the use of ephemeral keys as well can ensure that our next message exchange sessions can be secure as well.

Bibliography

[1] Dimitrios Desyllas. *Himitsu Project: Privacy for your mom*. 2018.

[2] W. Diffie and M. Hellman. "New Directions in Cryptography". In: *IEEE Trans. Inf. Theor.* 22.6 (Sept. 2006), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638. URL: http://dx.doi.org/10.1109/TIT.1976.1055638.

[3] Mike Burmester and Yvo Desmedt. "A secure and scalable Group Key Exchange system". In: *Information Processing Letters* 94.3 (2005), pp. 137 –143. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2005.01.003. URL: http://www.sciencedirect.com/science/article/pii/S002001900500013X.

[4] Ueli M. Maurer and Stefan Wolf. "Diffie-Hellman Oracles". In: *Advances in Cryptology — CRYPTO '96*. Ed. by Neal Koblitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 268–282. ISBN: 978-3-540-68697-2.

[5] Ahmad-Reza Sadeghi and Michael Steiner. "Assumptions Related to Discrete Logarithms: Why Subtleties Make a Real Difference". In: *Advances in Cryptology — EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 244–261. ISBN: 978-3-540-44987-4.

[6] Stefan Wolf. "Information-theoretically and computationally secure key agreement in cryptography". PhD thesis. ETH Zurich, 1999.

[7] "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels". In: Innsbruck (Tyrol), Austria, 2001, pp. 451–472. URL: https://www.iacr.org/archive/eurocrypt2001/20450451.pdf.

[8] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. "Stronger Security of Authenticated Key Exchange". In: *Provable Security*. Ed. by Willy Susilo, Joseph K. Liu, and Yi Mu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–16. ISBN: 978-3-540-75670-5.

[9] Moxie Marlinspike. *Simplifying OTR deniability*. 2013. URL: https://signal.org/blog/simplifying-otr-deniability/.

[10] Katriel Cohn-Gordon et al. *On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees*. Cryptology ePrint Archive, Report 2017/666. https://eprint.iacr.org/2017/666. 2017.

[11] *MLS Protocol Draft*. 2018. URL: https://github.com/mlswg/mls-protocol/blob/master/draft-ietf-mls-protocol.md.

[12] Mike Burmester and Yvo Desmedt. *A Secure and Efficient Conference Key Distribution System (extended abstract)*.

[13] P. van Oorschot A. Menezes and S. Vanstone. "Key Establishment Protocols". In: *Handbook of Applied Cryptography*. CRC Press, 1996.

[14] Michel Abdalla et al. *Flexible Group Key Exchange with On-demand Computation of Subgroup Keys*. Ed. by Tanja Bernstein Daniel J.and Lange. Berlin, Heidelberg, 2010.

[15] eQualit.ie. *(n+1)sec protocol specification — draft*. 2017.

[16] *Improved Group Off-the-Record Messaging*.

[17] Jens-Matthias Bohli and Rainer Steinwandt. "Deniable Group Key Agreement". In: *Progress in Cryptology - VIETCRYPT 2006*. Ed. by Phong Q. Nguyen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 298–311. ISBN: 978-3-540-68800-6.

[18] Matthew Van Gundy. *Improved Deniable Signature Key Exchange for mpOTR*. 2013.

[19] Qingfeng Cheng and Chuangui Ma. "Security Weakness of Flexible Group Key Exchange with On-Demand Computation of Subgroup Keys". In: (Aug. 2010).

[20] Alex Balducci, Jake Meredith, and Andy Lee. *(n+1)sec Cryptographic and Protocol Review*.

[21] Richard Barnes et al. *The Messaging Layer Security (MLS) Protocol*. Internet-Draft draft-ietf-mls-protocol-03. Work in Progress. Internet Engineering Task Force, Jan. 2019. 45 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-03.

[22] David McGrew. *An Interface and Algorithms for Authenticated Encryption*. RFC 5116. Jan. 2008. DOI: 10.17487/RFC5116. URL: https://rfc-editor.org/rfc/rfc5116.txt.

[23] Moxie Marlinspike. *Advanced cryptographic ratcheting*. 2013. URL: https://signal.org/blog/advanced-ratcheting/.

[24] *The Double Ratchet Algorithm*. URL: https://signal.org/docs/specifications/doubleratchet/.

[25] Ian Goldberg et al. *Multi-party Off-the-Record Messaging*. 2009.

[26] eQualit.ie. *(n+1)SEC*. 2016. URL: https://web.archive.org/web/20171116024305/https://learn.equalit.ie/wiki/Np1sec.

[27] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: Accountable Anonymous Group Messaging". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. Chicago, Illinois, USA: ACM, 2010, pp. 340–350. ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866346. URL: http://doi.acm.org/10.1145/1866307.1866346.

[28] Sarah Jamie Lewis. *Cwtch: Privacy Preserving Infrastructure for Asynchronous, Decentralized, Multi-Party and Metadata Resistant Applications*. Internet-Draft.

[29] *Overview*. URL: https://github.com/ricochet-im/ricochet/blob/master/doc/protocol.md.

[30] WHATSUP. *WhatsApp Encryption Overview*.

[31] Dr. Mike Pound. *What's Up with Group Messaging?* URL: https://www.youtube.com/watch?v=Q0_lcKrUdWg.

[32] *Threema Cryptography Whitepaper*.

[33] *The TokTon project*. URL: https://toktok.ltd.

[34] *The TokTok Project - Protocol*. URL: https://toktok.ltd/spec.html.

[35] Emad Omara et al. *The Messaging Layer Security (MLS) Architecture*. Internet-Draft draft-ietf-mls-architecture-01. Work in Progress. Internet Engineering Task Force, Oct. 2018. 16 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-mls-architecture-01.

[36] Peter Saint-Andre. *XEP-0045: Multi-User Chat*. Internet-Draft 1.31.2. XMPP Standards Foundation, 2018.

[37] E. Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Pearson Education, 1994. ISBN: 9780321700698. URL: https://books.google.gr/books?id=6oHuKQe3TjQC.

[38] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*. RFC 6121. Mar. 2011. DOI: 10.17487/RFC6121. URL: https://rfc-editor.org/rfc/rfc6121.txt.

[39] Lance Stout, Jack Moffitt, and Eric Cestari. *An Extensible Messaging and Presence Protocol (XMPP) Subprotocol for WebSocket*. Tech. rep. 7395. Oct. 2014. 18 pp. DOI: 10.17487/RFC7395. URL: https://rfc-editor.org/rfc/rfc7395.txt.

[40] Ian Paterson et al. *XEP-0206: XMPP Over BOSH*. Internet-Draft 1.4. XMPP Standards Foundation, 2014.

[41] URL: https://en.wikipedia.org/wiki/Unikernel.

[42] *Signal » Download Signal*.

[43] URL: https://www.recred.eu/.

[44] Dr. Mike Pound. *Instant Messagind and the signal protocol*. URL: https://www.youtube.com/watch?v=DXv1boalsDI.