Universität Konstanz
Department of Computer and Information Science
Distributed Systems Working Group

Master Thesis

Design and Analysis of a secure multi-party communication protocol

Klaus Herberth

September 6, 2016

1. Evaluated by Prof. Dr. Marcel Waldvogel
2. Evaluated by Prof. Dr. Hanno Langweg

Advisor: Matthias Fratz

Abstract

In the past years digital communication became an important aspect in every day life. Everything is shared and discussed in groups of friends, family or business partners without a proper way to protect that information. This master thesis introduces the first secure robust multi-party communication protocol which mimics a physical conversation with the help of a Diffie-Hellman key tree and social behaviours. Robustness against offline group members is reached by taking advantage of transitive trust between people and the ability to decline new members through an implicit protest. After introducing and analysing this new protocol, an improved version, which fixes the found issues, is presented.

Contents

1. Introduction

1.1. Why encrypted group communication?

In the past years communication became an important aspect in every day life. Everything is shared and discussed with friends and family. But not only in private section communication has changed, in business environments similar observations can be made. More and more important and sensible topics are discussed in realtime with group chats or video conferences without a proper way to protect the information given from eavesdropping. While there are some well studied protocols, like OTR messaging[1], for one-to-one conversations, there is no suitable solution for group communication.

Such a protocol should provide the same properties as a real group meeting. For example it could be the case that not all members know each other, depending on the level of confidentiality and group size. Most of the time it is enough to know that a trusted partner is acquainted with the other participants, or you get to know them before the actual discussion starts. That somebody is present also does not necessarily mean that this person pays attention. Usually it is sufficient to know the context of a statement, compared to knowing that a participant got every detail of the conversation. E.g. if a person answers, it is enough to know to which question he refers. Another aspect of a real group is that if a member broadcasts false or malicious information, all other members could potentially protest. Additionally a meeting should be protected from eavesdropping and statements should be deniable to non-members.

In the remaining chapter we take a deeper look at the properties and outline why no other protocol meets these. In chapter 2 we discuss and introduce our solutions to these problems. In chapter 3 of this thesis we present our new protocol, called Tree-based Group OTR Messaging (TBG-OTR) and analyse it in chapter 4. The improved protocol is presented in chapter 5. How this could be further improved is described in the first section of chapter 6. We conclude in section 6.2.

1.2. Why another protocol?

There is already some research in the field of secure group communication, but all attempts have some drawbacks as described below.

Cryptocat The open source browser extension Cryptocat uses pairwise messages to enable encrypted group conversations [16]. These distribution lists have the advantage that the server does not have to know about the concept of a group and therefore this solution is easy to implement and maintain. On the other side this results in great overhead and a lot of computations, because every message is individually encrypted for every member. For an attacker it is easy to discard messages and to produce misunderstandings between participants, as there is no consensus about transfered messages and a verification of the member list is not possible. Statements could be deniable depending on the used encryption protocol.

Signal The developers of the open source mobile messenger Signal are using the same approach as Cryptocat, but to save bandwidth they decided to encrypt messages with a symmetric key and only to encrypt this key for every user individually [31]. Besides this fact both solutions have the same drawbacks in consensus of messages and members.

Group OTR Bian, Seker and Topaloglu [7] introduce a virtual server to extend two-party OTR to a multi-party version. Obviously this concept is vulnerable to network failures and depends on the trustworthiness of a single user, which is not desirable.

Multi-party OTR Goldberg, Ustaoglu, Gundy and Chen [22] proposed a method of group communication which provides plausible deniability, confidentiality and authenticity. To achieve these properties they use three phases: Setup, Communication and Shutdown. The major drawback of this system is that all members of a group have to run these phases one after another and if messages get lost or manipulated, they get only notified in the very last step without the information what exactly went wrong. In lengthy meetings the shutdown phase would give feedback over a huge amount of messages and an attack would perhaps be discovered too late. Like the previously introduced Group OTR, this approach is vulnerable to network failures.

Improved Group OTR In their paper "Improved Group Off-the-Record Messaging" [29], Liu, Vasserman and Hopper tried to avoid the shutdown phase. For that purpose they use cyclic keys and chat digests, which scale badly for larger groups and require active participation to generate a common secret.

In the next section, we describe a protocol which avoids all these disadvantages by relaxing the requirements. As a result we got a protocol which is robust against network failure and absent users, while protecting from misunderstandings caused by false context, and verifiable statements.

1.3. Idea and Properties

The aim of a secure group communication protocol is to mimic a physical conversation in the digital world. To do so, we have to take a deeper look at the properties and behaviours of such an interaction.

In the real world the content of a group meeting, if we assume that no interception devices were used and that the meeting took place at a private area, is only known to the participants. Depending on the group size every participant can comprehend the source of every statement. In small groups this is easier, because the number of potential sources is limited and recognising voices is easier. In larger groups the allocation of voices to persons is more difficult and requires sometimes an enquiry. In meetings it is common that people get distracted and that even though they are physically present, they don't pay attention to the current speaker. For this reason, important announcements require always a receipt by e.g. show of hands or nodding. Many discussions have a dynamic group constellation with joining or (temporarily) leaving members and depending on the level of confidentiality of the discussed topic the admission process differs. Besides open discussions without any confidential content[1], for most everyday meetings transitive relationships are enough. Let us assume a group of people talking to each other. Now if a new member, lets call him Jon, wants to join, there exist two scenarios: Somebody in the group knows Jon and introduces him to the group and all existing members accept him. If nobody knows Jon, he has to introduce himself and group members get to know and vote for him. In both cases all old members have the chance to leave the group or to protest against Jon after he joined. A user can in the best case announce his leave or just silently leave. This last case shows that a receipt on important statements is very important. Two other properties of group discussions are that every participant can contribute equally[2] and that nobody can prove participation or statements to third parties. This kind of deniability protects only against a dishonest member, but not against the testimony from the collective, which is also known as the fundamental problem of deniability (FPD).

A digital meeting should mimic all these behaviours, which results in the following properties for our protocol: It has to be **encrypted** to prevent third parties from listening. **Message authentication** is important to get the source of every statement. **Identification** has to be provided in order to recognize persons. Our protocol has to handle **group events** like joining or leaving, and treat every member **equally**. Participation and course of conversation should be as **deniable** as possible. Additionally it should be **robust** against not responding members e.g. if they have temporarily no internet connection, it should still be possible to add new members. One more helpful property, which would be desirable also in real life, but is only realizable in digital communication, is having a way to **reference** a specific message. Especially in large

[1] e.g. discussions about new video games, weather or movies.

[2] All members have the same possibilities to join, leave or speak, even if there is some kind of moderator or policy.

groups this would help to keep the overview. The following security properties should also be achieved by a secure group communication protocol, as described in [24, 27]:

Perfect Forward Secrecy (PFS) Compromising a long-term secret should not result in the compromise of old discussions.

Group Key Secrecy A passive adversary should not be able to guess or calculate the group secret.

Forward Secrecy An adversary who knows group secrets should not be able to compromise following secrets.

Backward Secrecy An adversary who knows group secrets should not be able to compromise preceding secrets.

Key Independence An adversary who knows group secrets should not be able to compromise any other group secret.

Similar to [27], a leak of secure session parameters should not influence other sessions in our protocol and therefore prevent e.g. replay-attacks [33].

In the next chapters we will see how these properties can be transformed into a secure group communication protocol.

2. Protocol Principles

In this chapter we will introduce the principles of the Tree-based Group Off-the-record (TBG-OTR) protocol and compare different solutions to realize them. The first section describes the key exchange between multiple parties, followed by a short description of the used encryption algorithm. Section 2.3 introduces a concept called implicit agreement, section 2.4 covers message authentication and section 2.5 participant awareness. The last section describes how misunderstandings in groups can be prevented.

2.1. Key Exchange

Key exchange is one of the major problems in designing a secure group communication protocol. It should serve the following properties:

- Authenticated
- Require only a few messages to add or remove a member
- Robust against not responding parties
- Every party should contribute an equal part to the resulting secret
- New or former members should not be able to compromise old respectively new keys (key independence)

There are many protocols, e.g. [13, 12, 14, 35, 37], which describe a Group Key Exchange (GKE) and with [23] it is possible to transform even the unauthenticated in an Authenticated Group Key Exchange (AGKE) protocol. But all these solutions require active contribution from all users. Augot et al. [3] present a solution which does not suffer from this limitation, but it requires more computations and a fresh D-H key from the former group leader for the next group event (join/leave).

TBG-OTR uses a variant of "Tree-based Group Key Agreement" [25] from Kim, Perrig and Tsudik, in combination with DSA signatures[1]. The basic concept is that every member is a leaf in a binary tree and that the value of a node is the result of a Diffie-Hellman key exchange [18] between those children. E.g. a node C has two children (A, B) with the values[2] $bk_A = g^a$ resp. $bk_B = g^b$. The result (shared secret)

[1] Using DSA signatures to transform this unauthenticated in an authenticated GKE protocol has the advantage, compared to [23], that it is still robust against not responding members.

[2] All exponentiations are done modulo a particular prime.

of a key exchange of A and B would be $k_C = k_{AB} = g^{ab}$ and therefore the blinded value for C would be $bk_C = g^{k_{AB}} = g^{g^{ab}}$ which is used in the next key exchange as public DH key (see figure 2.1). If all blinded keys in a tree and one corresponding private key are known, a root secret can be calculated. Since every member is a leaf, they have also the private part of the stored public key and are therefore capable of generating the group key if all members share the same tree. Obviously this scales well for large groups, because every new member has to calculate only tree height[3] (h) many DH keys if they join. All existing members have to calculate in average less than 2 DH keys for a perfectly balanced tree.

$$2 \approx \lim_{h\to\infty} \frac{1}{n} \cdot \left(\frac{(h+1)\cdot n}{2^h} + \sum_{i=1}^{h} \frac{i \cdot n}{2^i} \right) \tag{2.1}$$

$$\approx \lim_{h\to\infty} \frac{h+1}{2^h} + \sum_{i=1}^{h} \frac{i}{2^i} \tag{2.2}$$

$$\approx \lim_{h\to\infty} \sum_{i=1}^{h} i \cdot 2^{-i} \tag{2.3}$$

See section 3.1 for building and sharing trees.

The security of this protocol is based on the assumption that the Diffie-Hellmann algorithm is secure, a full proof was done in [2].

2.2. Encryption

Secure group communication requires strong encryption to prevent eavesdropping. A common algorithm is the Advanced Encryption Standard (AES) [20], which is used by many applications e.g. OTR, and besides some recently proposed attacks [11, 8, 19] still presumably unbreakable [38].

TBG-OTR uses AES in counter-mode with a key length of 128, because it needs less computing power and as Biryukov et al. [9] showed longer keys are easier to break under certain conditions.

2.3. Implicit Agreement

A group has to agree on different topics, like the joining of a new user or the correctness of the current group parameters. While some agreement protocols (Two-phase commit [6], Three-phase commit [34]) require active participation of all members, the Paxos

[3] In a totally unbalanced tree this would be at maximum $|members|-1$ and for an perfectly balanced tree $log_2(|members|)$.

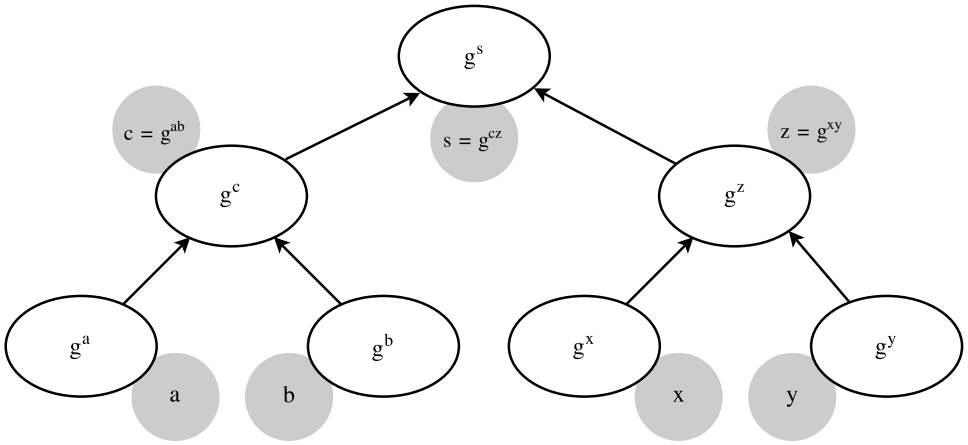


Figure 2.1.: The value of a node consists of the result from a Diffie-Hellman key exchange done with a known private (gray background) and available public key (white background) of two children. Every leaf is linked to a member which owns the private part and is therefore capable to compute the root secret by computing all DH key exchanges up to the top.

[28] concept allows to handle even not responding members. Nonetheless all agreement protocols require a lot of messages, which can be avoided with implicit agreement.

For example, all existing members agree on a new user by using the new key which enables the user to read that message. In this case an explicit agreement is not necessary, because the user who would disagree with the new member can only control which messages he sends. If he thinks this user is not trustworthy, he has to give reasons anyway. As a result we can say that such issues should be discussed on a human and not on a protocol level. Another disadvantage of explicit agreement on new members, would be that each member needs to confirm a new member, which can be annoying and time consuming in active groups. As a result people would claim an auto-confirmation option from their client, which results in exactly the same as with the implicit agreement approach.

As we try to avoid pairwise messaging, we introduce a new action called protest, to avoid explicit agreement on group parameters, proposed identities and other group information. If we assume that a group which decided to cheat us as a whole, is always capable of doing so, we also have to assume that one honest member is enough to protect us from frauds and that this person would protest against misinformation. If we receive no protest, we interpret this as agreement over all leaves and therefore members.

2.4. Message Authentication

For every member it is important to determine the origin of a message to prevent misunderstandings. Goldberg et al. [22] use ephemeral signing keys, which get exchanged pairwise and published afterwards. This approach is, as outlined in section 1.2, not robust against not responding members. To get deniable message authentication, we use the fact that if the number and identity of all participants is known, the number of message origins is limited. A honest member would protest (section 2.3) if someone tries to trick a new member or the use of multiple signing keys would be recognized. For this reason every member signs his messages with a short-lived Digital Signature Algorithm (DSA) [30] key and new members trust those keys on first use, if nobody protests, compared to the pairwise exchange from [22]. It is important to stress that this short-lived DSA key is never verifiable through the users long-lived key and verification is only done through a deniable channel, like OTR, to ensure deniability of all signed messages.

The Digital Signature Algorithm [30] was chosen as preferred signature algorithm, because it produces shorter signatures compared to RSA [32], but TBG-OTR includes a flag to support different algorithms in the future.

2.5. Participant Awareness

In the previous section we assumed that the number of listeners and therefore participants is limited. To recognize people and link messages with their origin, every participant has to know who else is part of the current group. As outlined in section 2.1, every member is a leaf in a common blinded key tree. Therefore if every member is capable of generating the group secret, the membership of all participants is proven. As a consequence all new members have to contact every old member to verify that they are part of the group. Since the DH value should change in every new group, we need another way to determine people behind their public keys. Similar to OTR we use DSA signatures to identify persons between multiple sessions. Every leaf has to be signed by the user with his long-lived DSA key to recognize him in the next meeting. In this way we do not need to verify every user if we know all members in a group[4].

2.6. History Awareness

The purpose of history awareness is to ensure that every participant knows the context of a statement, instead of ensuring that every member got each message[5]. This

[4] If confidential topics are discussed, we recommend to verify every user, as he could be impersonated.

[5] This concept of receipts is usually used by other multi-party protocols (see 1.2) to reach consensus about all messages, but as outlined this is not needed.

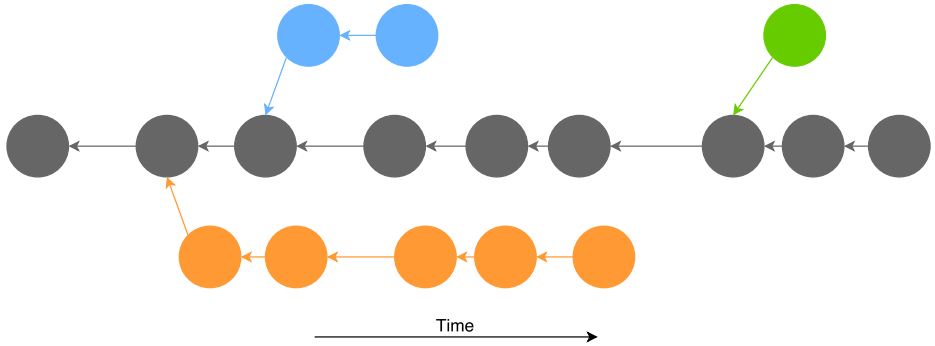


Figure 2.2.: Every message (circle) has exactly one parent message, which allows branches, but no merges, like in the git tree.

behavior mimics a physical meeting, in which nobody can guarantee that everyone focuses on the discussion. E.g. after an important announcement, it is common to wait for an acknowledgement and not to assume it. For that reason every message (except the first one) has to contain a hash of the corresponding or last received message. In this way every person knows to which message an answer belongs and it is even possible to reference a certain message, which can be helpful in large discussions. This results in a tree (figure 2.2) similar to the git [15, Chapter 10] history in which every commit depends on all related commits before. Therefore, similar to git, every message authenticates many messages before and additionally conversation branches are possible, which is common in group communication.

3. Solution Design

In the next sections we will describe how a new member joins an existing group, how they exchange data and how participants leave a group. The following assumptions are made:

- We require a broadcast channel for all actions which delivers messages for all participants in the same order.
- For all Diffie-Hellman group computations, we use the 1536-bit MODP Group from [26] and a generator (g) of 2.

3.1. Joining

If a new user (M_{n+1}) wants to communicate securely, he broadcasts a special[1] plaintext message (S1 in figure 3.1) to detect if there is an existing group (C) with n users. If this is the case, the last active member responds with a TBG-OTR error message (S3). As a result M_{n+1} broadcasts a request (figure 3.3) with his blinded key (public DH key, bk_{n+1}) together with its signature, produced with his long-lived DSA key (S5). Every member $M_i \in C$ checks this signature and sends a confirmation if he knows M_{n+1}. If the number of confirmations is greater than the configured limit, the rightmost shallowest member in the key tree, called sponsor[2] (M_s), starts calculating the new key tree (see figure 3.2). This is done using the following steps:

1. Delete all blinded keys (bk) on the way up to the root.
2. Add bk_s and bk_{i+1} as children to the previous position of M_s.
3. Calculate all blinded keys up to the root, as described in 2.1.

The resulting tree, containing only blinded keys and serialized in pre-order[3], is broadcasted together with a list[4] of all members to $C \cup M_{n+1}$ (S7; figure 3.4). See figure

[1] Any message (even empty) starting with `?TBG-OTR?`.

[2] The need of a sponsor does not harm the security of the system and does not weaken the system. (See section 4.4.)

[3] Pre-order traversal has the advantage that the resulting serialized tree is as small as possible and can easily be unserialized without buffering large parts of the tree in memory.

[4] The list contains signatures of all leafs generated with the long-lived DSA key of the corresponding member.

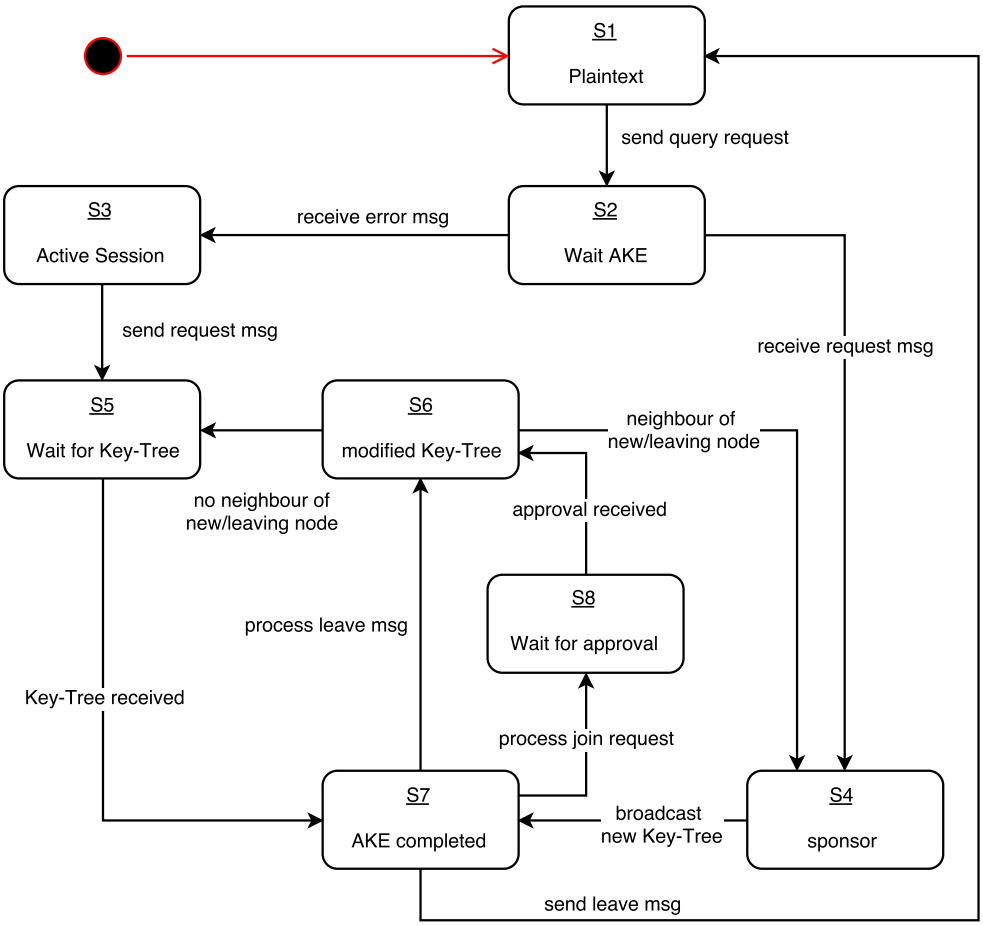


Figure 3.1.: This diagram visualizes the different states a user client has to handle from joining till leaving a group.

3.5 for a visualization of the overall sequence. All members who agree with M_{n+1} calculate all DH keys on the way up to the root from their node as described in 2.1 and use from that point on only the new secret. If they don't agree, they use the old keys. The new member has to trust that he received the correct tree and that otherwise someone would have protested, or he has to verify that information with pairwise messages to a subset of all declared members.

This procedure does not harm the security of the protocol at all, because it influences only the assumption of online members. If a user says something important which he wants everybody to know, he requires acknowledgement similar to a physical nod. In the other case if the new member says something confidential, he has to trust every member, which includes the sponsor, because otherwise a malicious person could abuse his trust. So if he does not trust the correctness of the tree, he does not

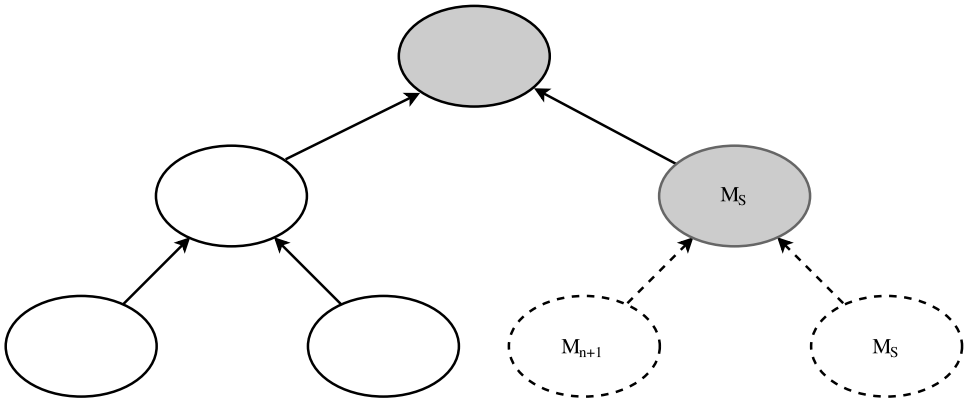


Figure 3.2.: In preparation for admission of a new user (M_{n+1}) all members have to delete the content of nodes from the rightmost shallowest member (M_s) up to the root (gray background color). M_s creates the new tree by moving his node one level down and adding the new node as his neighbour. Now he calculates all keys up to the root as described in 2.1 and publishes the blinded keys to the new group.

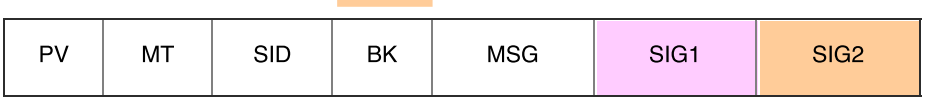


Figure 3.3.: This is the message structure of the join request. It consists of protocol version (PV), message type (MT), sender id (SID), blinded key (BK), message (MSG), DSA signature, using the long-lived DSA key of everything before (SIG1) and signature of BK (SIG2) which can be transfered as part of the tree update.

trust the sponsor and should therefore be very careful with his statements.

3.2. Exchanging data

This section outlines the method used to protect the data being send from M_i to all other members C. Data could be a plaintext message (msg), a disconnection request, a protest message[5], a public DSA key or a combination[6] of all. All this data is always signed with a short-lived DSA key in combination with the last received or related

[5] A protest message has to contain a reason and the id of the message which triggered the protest.
[6] This flexibility is reached through the use of Type-Length-Value (TLV) records. Each item consists of a specified number and the length of the following data.

Figure 3.4.: This is the message structure of the key update. It consists of protocol version (PV), message type (MT), sender id (SID), receiver id (RID), previous receiver id (PRID) which is used to link the old and new group, group configuration (CONFIG), counter (CTR), key tree (TREE) serialized in preorder, list of leaf signatures (SIGS), DSA signature, using the short-lived DSA key of everything before and message authentication code (MAC) of everything before.

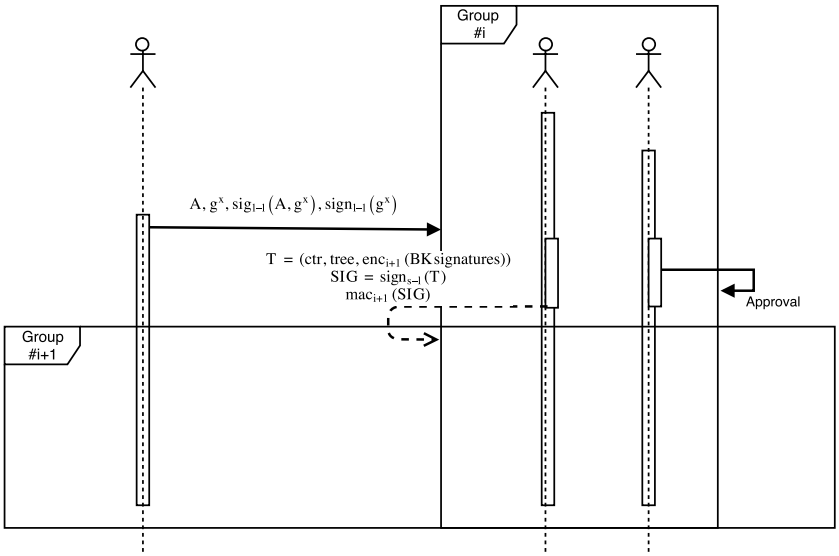


Figure 3.5.: After the user sends his join request and another member approves him, the sponsor sends the new key tree to all members (including the new user).

message (history). (See figure 3.7 for the structure of a data message.) To generate such a packet M_i has to process the following steps (figure 3.6a):

1. Select the key tree by reference to the desired participants. This means a message could theoretically also be send to a sub-tree of the main key tree.
2. Use the key tree to generate a shared secret (s) for C, and compute[7] the sending AES key (ek) and the sending MAC key (mk).
3. Pick a value of the counter (ctr) so that the tuple (s, ctr) is never the same for more than one data message.
4. Generate signature (sig) of (history + msg) with short-lived DSA key.
5. Compute T = (ctr, AES-CTR$_{ek,ctr}$(history + msg + sig)).
6. Send T, MAC_{mk}(T).

Every $m \in C$ generates the same shared secret s, receiving AES key ek and MAC key mk, if they receive a data message (T). In order to get the value, they have to (figure 3.6b):

1. Verify the MAC.
2. Decrypt the message with ctr and ek.
3. Check the signature with the first received and used public DSA key of that user.
4. Review related message to check if it exists.

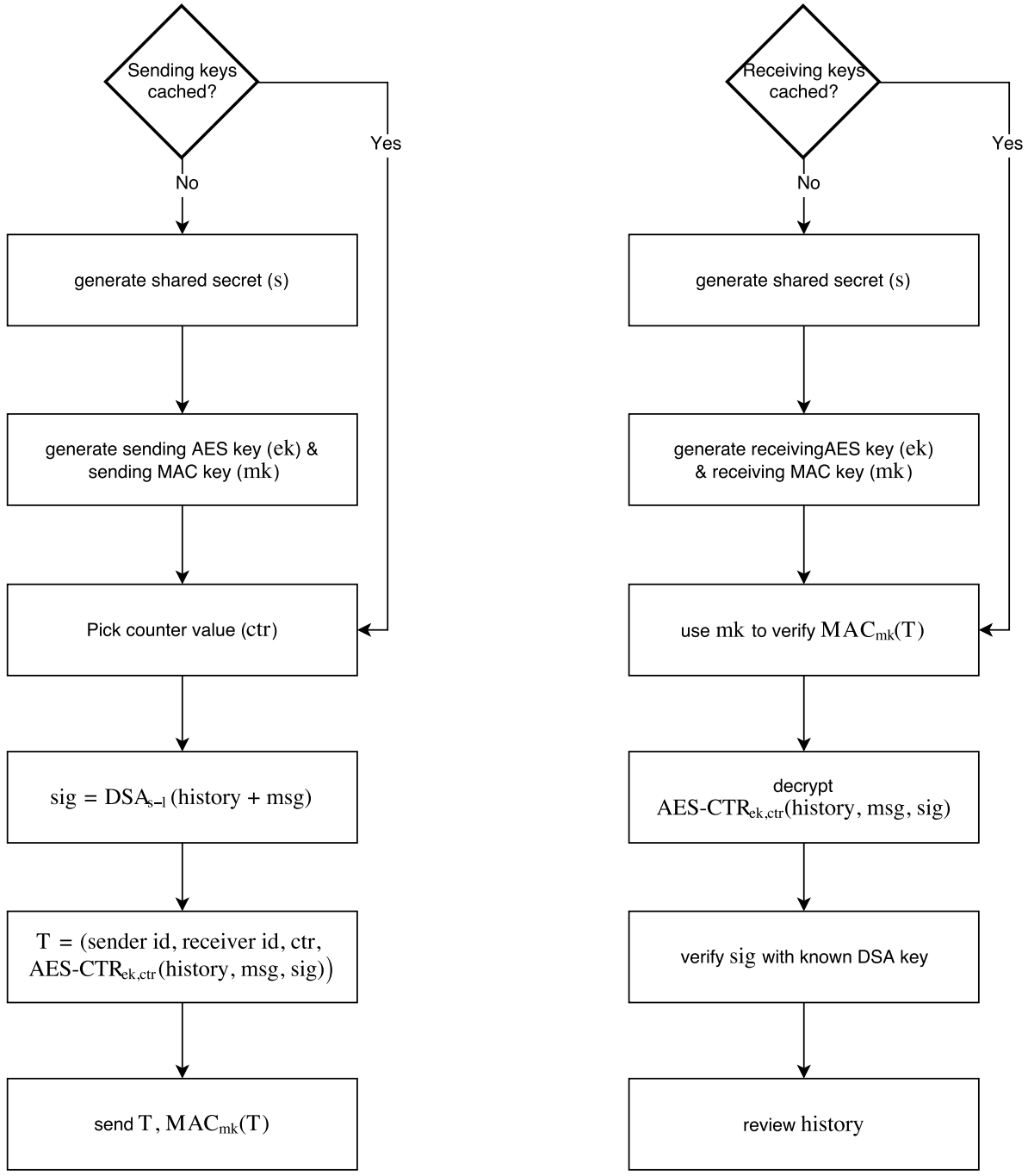
If one test fails, the message should be discarded and in the last two cases a protest needs to be sent.

All encryptions are done with AES in counter mode and a key length of 128-bit which is currently not breakable and requires fewest computing power.

3.3. Leaving

If a user M_d leaves a group, by sending a data message with a leave record, all other members $C \setminus M_d$ should update their key tree by removing the leaving member node and relevant parent nodes including their keys and blinded keys (see figure 3.8). The former neighbour M_s of M_d has to compute and broadcast a new tree with all public DH keys on the key-path. All members should compute and use the new group secret resulting from the updated tree as described in 2.1.

[7] For a given salt, define h1(salt) to be the 160-bit output of the SHA-1 hash of the salt followed by s. ek is the first 16 bytes of h1(own public DH key id). The sender id is used as salt to exacerbate cracking. mk is the SHA-1 hash of ek.



(a) Sending process **(b)** Receiving process

Figure 3.6.: Processing outgoing or incoming messages for a specific group requires multiple steps.

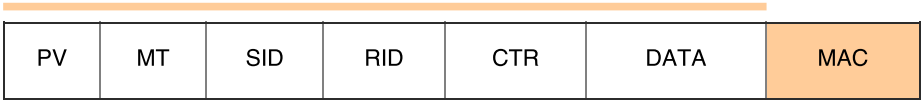


Figure 3.7.: This is the structure of a data message. It consist of protocol version (PV), message type (MT), sender id (SID), receiver id (RID), counter (CTR), payload (DATA) and message authentication code (MAC) of everything before. The payload consist of a reference to a message, a plaintext message (even empty), zero or more TLV records and a signature with the short-lived key of the first 3 items.

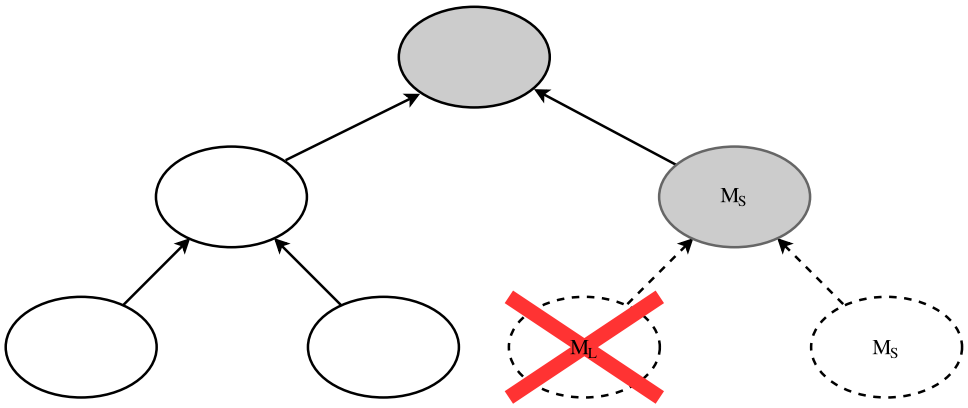


Figure 3.8.: After a user (M_L) left the group, all members have to delete the content of all nodes from M_L up to the root (gray background color), remove the node from M_L and level his neighbour M_S up. M_S calculates all values up to the root and broadcasts the blinded keys to the new group.

4. Analysis

In the following sections we will analyse if TBG-OTR violates any of the properties defined in section 1.3. Therefore we define different types of adversaries in the first section and look at different attack points in the following sections.

4.1. Adversaries

In this section we will introduce different types of adversaries and explain their abilities and goals. An insider is always part of the group, while an outsider is not. Active adversaries are able to modify, suppress or replay messages, compared to passive adversaries who are only able to read messages.

The passive outsider tries to learn as much as possible about the group from reading the exchanged transcript. However an active outsider can weaken the entire protocol or produce misunderstandings through altered messages. Compared to a passive outsider, a passive insider tries to accumulate evidence to convince another non-member that certain statements were given within the group or someone participates in the group. Additional to this goal an active insider tries to spoof other members about the origin of a message or the identity of other participants.

4.2. Information Disclosure

In the following section we will look at the information an attacker can get from the protocol transcript. While he is not able to modify or suppress any messages, he tries to get as many details as possible. Under the assumption that AES is theoretically secure (see [38]) and remains despite different attacks (e.g. [8],[11],[19]) even practically so today, an attacker can only get information[1] from the key exchange. As is obvious from figure 3.3, he can read an optional message which a joining user has send with his join request. This message could in some circumstances reveal important information about the user and group. For example the topic or even the identity of some or all members could be compromised and this information is even verifiable, because it is signed with the long-lived key of the joining member. A possible evidence for participation in a specific group is the signature of the public Diffie-Hellmann key in

[1] Not considering information about message transfer, response time and other data which is usually revealed in electronic communication.

combination with the exchange of the key tree in which this key appears as a leaf node. The attacker can not prove participation, because he is not able to calculate the root secret and verify the correctness of the tree.

Both information disclosures could be avoided by exchanging the signature only over an encrypted channel, similar to the SIGMA protocol [27]. In this scenario the joining user first broadcasts his blinded key to the group and the sponsor adds him immediately without knowing anything about the new user. Next the user generates a root secret and encrypts his signatures in the same way as usual messages. More confidentiality is achieved, if the sponsor encrypts also the key tree with the current group key and additionally[2] with the Diffie-Hellman (DH) secret of him and the new member. One major drawback is that all members have to be extremely careful about which user has been approved and which hasn't. Another drawback is the handling of concurrent joins, because in this case we either have to wait that a new user is approved before[3] another is added or different trees exist for every user which tries to join the group. In the last case the tree, for which a user gets approved, would be used, while all substitute trees have to be deleted, so that the user has to send a new join request. Both options seem to be error prone and compared to the grade of information disclosure too complex.

The mentioned tree message does not only contain the unencrypted key tree, but also the unencrypted group options. In certain circumstances these options could simplify an attack, but in the current version the possible values are very limited and therefore an attack is very unlikely.

4.3. Message Transfer

In the previous section we looked at the possibilities of a passive outsider. Now we take a look at the capabilities of an attacker which is able to suppress, replay or change messages. We focus only on such attacks which result in misconceptions or disadvantages for members, because Denial of Service (DOS) is always possible and therefore not mentionable.

Block ciphers like AES are generally vulnerable to Bit-flipping attacks, which enables attackers to alter a specific message without the need to decrypt it. In our protocol e.g. an attacker could lower the version number to exploit an older flaw. To protect against it, every message (except the join request) comes with a Keyed-Hash Message Authentication Code (HMAC) which enables every member to detect such changes.

One of the greatest issues in designing a group communication protocol is consensus about discussed topics. An attacker could suppress, delay or replay messages to a

[2] In practice the key tree would be encrypted with a key which is itself encrypted by the group key and the DH secret to save bandwidth.

[3] Otherwise the second maybe has to rejoin, because he is below the first user in the tree.

subset of members to produce serious misunderstandings. It is left up to the reader to think about different scenarios. Our proposed protocol uses 3 different kinds of defensive measures to protect users from such threats. To prevent an attacker from delivering packets between groups, all data packets contain a receiver address and are only readable for the corresponding group. Additionally all messages are linked to each other in order that every member can comprehend which answer belongs to which question. This reference prevents also from misuse[4] of the signed block by another member.

4.4. Key Exchange

The authenticated key exchange is the most important and critical task in designing a secure group communication protocol. As many protocols before (e.g. OTR version 1 [17] or STS [10]) TBG-OTR suffers from various flaws.

TBG-OTR uses signatures from long-lived DSA keys to recognize people during multiple group sessions, but the required public key is never transmitted. It was assumed that this key is passed between users during verification. It was not considered that people start to trust other people if they meet them more often and in this case an identity verification is useless[5], because they don't know each other in person. To resolve this issue we could extend the protocol to publish also the public DH key, similar to the short-lived key, with the first message after a new member joined.

According to our security properties, the protocol should only be breakable if you are in possession of a trusted long-lived key. This is indeed not the case, because an attacker who knows a corresponding secret DH parameter, could reuse an old join request. To prevent such attacks the signature should contain "a random or non-repeating value" [33] (nonce), a timestamp, or other information which proofs the destination of the request. While including a nonce requires an additional message, timestamps suffer from practical issues as seen on the example of Kerbos [5]. Therefore the use of the receiver id[6] does not only protect against replay attacks, but also helps to ensure that a user does not join every group in a broadcast channel[7].

The Unkown Key-Share (UKS) attack, as defined by Blake-Wilson and Menezes, describes a way in which person A believes he shares a key with person E, but in reality he is talking to B. On the other side B correctly assumes that he shares the key with A. Although E is not able to read the communication, he could be the beneficiary if identification is only done via the long-lived DSA key. For example, B takes part in a quiz game and wins the first price. The organiser transfers the money to the associated account of E, because he thought E was playing. Like figure

[4] e.g. replay the same signed block with a new message.

[5] To agree upon the group parameters, verification is still needed.

[6] The receiver id is the fingerprint of the root node.

[7] Due to the design of TBG-OTR multiple groups in one broadcast channel are likely. See section 4.6 for more information.

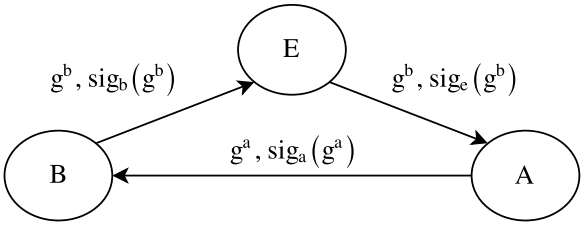


Figure 4.1.: For an UKS attack an adversary E has to replace the original signature from B with his own. As a result A assumes E as discussion partner.

4.1 illustrates, the attacker E replaces for this fraud the original signature from B with his own and forwards it to the intended receiver. In the case of TBG-OTR, the member list[8] protects against this attack from an active outsider, because he is not able to modify this list of signatures before it arrives at B together with the key tree. Whereas a malicious group member could intercept the tree update and replace his signature with the original one from B. In doing so, B has no possibility to detect that everyone else thinks he is E. To solve this issue, every member could attach a hash value of the member list to every message signed with his long-lived key, or two-party verification of as many members as possible is required. (See the next chapter for further discussion.)

Special roles, like the sponsor, always need to be investigated, because they are likely to weaken the entire protocol. The sponsor is able to modify the key tree and add members below his leaf node, therefore it is important that all members compare the new tree with the latest tree and protest if they detect any unauthorized change. This is clearly not possible for hidden members below the sponsor node, but it has the same effect as if the sponsor would publish the secret key to an eavesdropper, therefore we have no possibility to prevent this flaw. As seen in figure 4.2, the sponsor could use a root secret from a group of eavesdroppers as his own DH parameter.

4.5. Message Authentication

One of the corner stones of message authentication is a secure hash algorithm, because it is used to calculate fingerprints, keys and signatures. Recently Stevens, Karpman, and Peyrin [36] discoverd the first practical collision attack on the full SHA-1 algorithm, which TBG-OTR is using. While our protocol only requires second-preimage resistance, attacks only get better and for this reason a newer algorithm like one of the SHA-2 or SHA-3 family should be used in the future. As Bellare [4] showed, the calculation of secure HMAC is not requiring collision resistance from the underlying hash algorithm, whereby further use of SHA-1 for MAC generation is currently not

[8] A list of all blinded leaf keys with their corresponding signature which was published together with the join request.

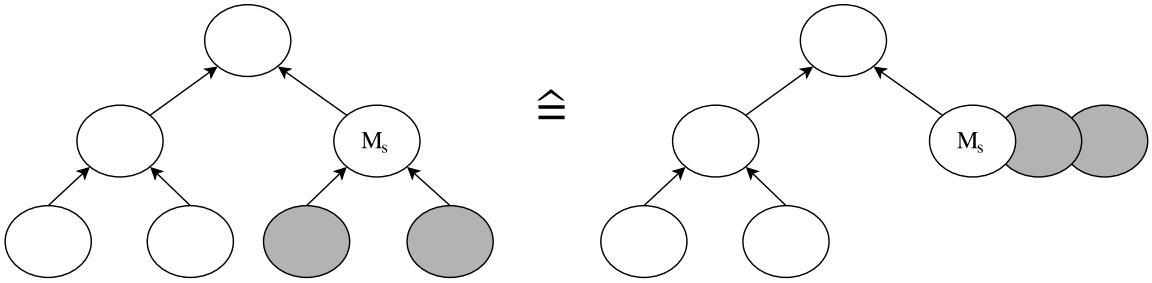


Figure 4.2.: The sponsor M_S has the possibility to add multiple adversaries (gray background color) under his node, but this would result in the same as he would share his group parameters. Both attacks require the same amount of trust from the adversaries, because they have no possibility to verify those shared secrets.

affected.

Truly deniable message authentication[9] in group communication is still an unresolved problem. TBG-OTR's used method provides only deniability until the owner of the short-lived DSA keys is publicly verifiable. This could happen accidentally through signing the short-lived key with the long-lived key or a signed statement which is only possible from a known person. Even publishing of used private keys, like OTR does with old MAC keys, does not improve deniability, because all messages are linked to each other and therefore it is extremely[10] difficult to forge a complete transcript.

The combination of a member list with old signed blinded keys from section 4.4 and the ability to impersonate all members, results in a possible Man-in-the-Middle (MitM) attack for an insider. It is important to stress that this is not possible for a person which is not part of the group, because otherwise no plausible messages could be generated. To perform such a fraud, an attacker needs signed blinded keys and has to construct a plausible, but even fictional, tree with those keys as leafs. The only requirement is that the attacker is the sponsor in this tree and knows the private parameter of the corresponding blinded key. This usually also means that he is in possession of the DSA key which was used to generate the signature of this parameter. (For an illustration see figure 4.3.) After the key exchange the new member believes he is part of a group with all persons the sponsor mentioned and nobody protests, because no other member exists. Every message the victim receives seems correct, because it is encrypted and authenticated correctly and the attacker can generate short-lived DSA keys for every imaginary member. To prevent such an attack, we could demand

[9] Even revealing information which is only known to a specific person does not influence past or future statements.

[10] This would require to know all short-lived private keys from a group, which would require a shutdown/end phase. This last step would make the protocol not robust against not responding members.

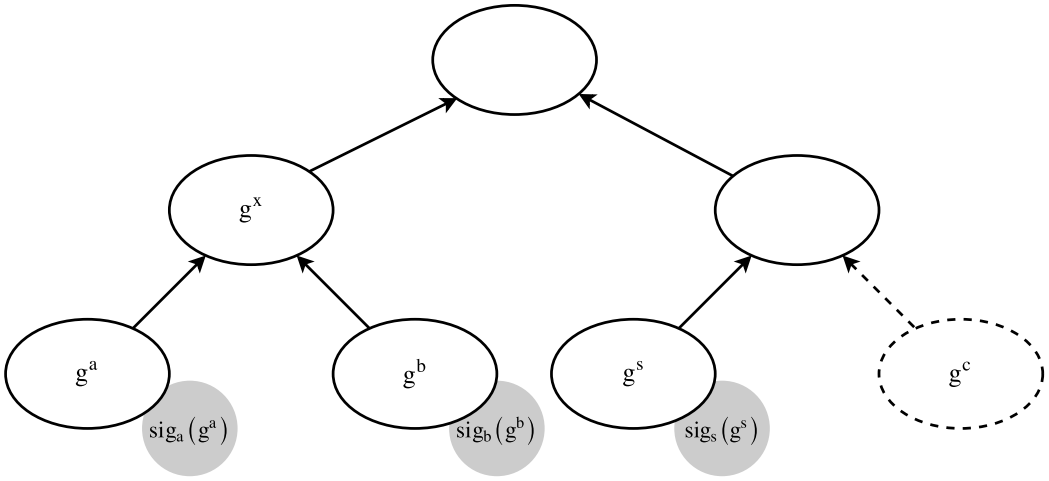


Figure 4.3.: If a user (C) with public DH key g^c gets added to a tree, he has to trust that the sponsor (S) sends him the correct tree, because C is only able to verify the path up to root. All other paths could be faked by S.

that with every message the author adds a signature with his long-lived key from the current tree. Similar to the SIGMA protocol [27], this would guarantee that the owner of that key is able to generate the secret. On the other hand the ability to impersonate the group results in higher deniability of participation. (See chapter 5 for further discussion.)

4.6. Conceptual Issues

This chapter covers protocol issues which do not exclusively require an attacker. They could occur because a member went offline or a message got lost.

4.6.1. Timeouts

On the protocol state machine (figure 4.4) we see 3 states which require information from other users and don't have an alternative exit. If the required user does not respond a deadlock occurs and for example a new user can not be added. In the following paragraphs we will look at each state in the order in which they could occur for a joining member.

The first timeout can occur after a user sends a query request message to either start a new group or detect one. This would mean nobody responds because nobody is online or the group does not want more people to join. Both cases do not harm the protocol and the user will dismiss his request after a short amount of time.

Next the user is waiting for the key tree to calculate the root secret and starts communicating with the group. If no existing member sends an approval or the

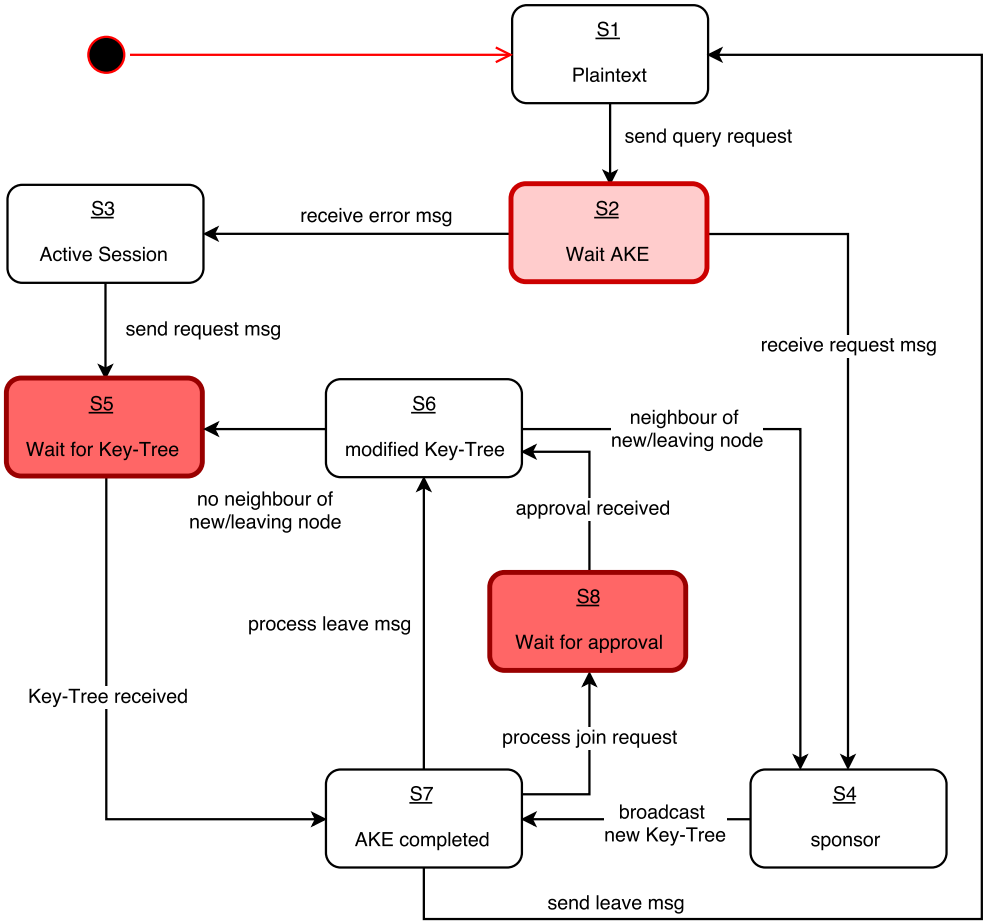


Figure 4.4.: In 3 states users have to wait for information from other members, but only S5 and S8 are critical. A user in S2 would abort his join request after some amount of time and it does not effect existing members.

sponsor is not able to calculate the updated tree (e.g. he is offline), the new member will stay in this state until the deadlock is solved or he quits. Compared to this new member it is much more problematic for existing members to stick in this state, because they are unable to add or remove users. The protocol defines for this case a fixed timeout of 8 seconds, but fails to provide any further information about how to proceed after this amount of time.

The protocol assumes that join requests are processed sequentially, resulting in a deadlock even if someone could approve the second user, because all users wait for the approval of the first request. To fix this, join requests should be processed in the order of received approvals. This message could also contain the current tree, which would enable the new member to work as sponsor and obsolete the requirement of an additionally responding member[11]. This does not influence the security of the protocol, because the key tree can be public information. For more security it could be encrypted, like described in section 4.2. To prevent a joining user to block other group events through not publishing the updated tree, a timespan should be defined after which the lock should be released in order to add other members. The not responding member should request entrance again, if there is a difference between his received and the currently used group tree.

4.6.2. Verification

TBG-OTR recommends verification in confidential meetings to review session parameters, like key tree or list of members, and the identity of the long-lived DSA key. The protocol leaks a detailed description of which parameters need to be verified and defines only rough requirements for that process. As seen earlier, verification of session parameters is also important to prevent MitM attacks. In any case the first part of verification, the session parameters, could be done fully automated to increase usability and overall protocol security. The aim is to verify that an owner of a long-lived key has the same view of the key tree and member list as the initiator of the verification. (For a solution to this issue, see chapter 5.) As outlined in section 2.5, the number of pairwise verifications depends on the level of confidentiality. The user interface should therefore provide a selection to define the kind of topic and visualize the number of sessions and key verifications.

4.6.3. Mass Joins and Leaves

As seen earlier TBG-OTR processes join and leave requests sequentially, which is good for durable groups with only a few group events at a time. By contrast in ephemeral groups, a lot of people join at a stroke which requires an explicit approval for every single user and therefore results in a long initial phase. This could be accelerated by using a divide and conquer (D&C) approach, whereby people are partitioned in

[11]The traditional sponsor.

smaller groups with their own sub-key tree. In this way joins can be parallelized and the merge of key trees is very efficient, so that this approach would result in much faster group initialization. On the downside a smart concept is needed which decides when to start the D&C process and how people are partitioned including an agreement over the participants. Deployment and evaluation of such a system is ceded for further work, see chapter 6.

Not only mass joins, but also multiple leaves in a short amount of time could create problems with the current specification, because it does not address the case of multiple leaves for one group id. To handle such a case it should be defined that the request should be processed for that group which results after all pending group events before the request was send were processed. Further it should be defined how to handle cases in which the sponsor does not respond (see figure 4.4) because he is offline, or also sent a leave message.

4.6.4. Protest

The approach of implicit agreement via protest has many advantages as seen in section 2.3, but in large groups this leads to a lot of messages at once which could overextend low-performance devices or devices with a low bandwidth. To avoid this issue, all members could delay the sending of their protest message by a random time within an interval up to some defined[12] seconds, similar to the concept of Floyd et al. [21].

Another issue is the implicit protest against a new member by continuing to use the previous group key without leaving the new group. This could lead to a timeout if this member would become the sponsor of the group which he doesn't want to be part of. Even the explicit protest via a leave message could cause issues if it occurs at a bulk as seen in section 4.6.3.

[12]The end of this interval could be a fixed number of seconds, or depending on the round-trip time.

5. Improved Protocol Version

In this chapter we look at changes which were necessary in order to fix issues from chapter 4 and discuss why not all proposals were realized. In the first section we cover the biggest change to the protocol in order to be more robust against not responding members. Section 5.2 introduces a group parameter verification process and the last section contains smaller, but no less important, improvements.

5.1. Key exchange

The state diagram from figure 4.4 shows that a deadlock occurs if nobody approves the new member or the sponsor does not respond. In the improved protocol version it is now allowed to process the next request after a reasonable amount of time.

> *If there is a reasonable time between two transitions and there is another pending join request, it is allowed to start processing the next request which automatically terminates the existing request. (D.3.7.2)*

It is not possible to define a specific duration, because it does not only depend on a round-trip time, but also on the computing power of the new member and the time an existing member needs to approve the candidate. For this reason we believe that while this delay depends on a human factor, a waiting time should also consider humans.

The need of a second responding member (sponsor), besides the approver, was eliminated by letting the candidate calculate the updated tree. This required a new message type (see figure 5.1) to announce the current tree and some adaptations to the tree update message (figure 5.2).

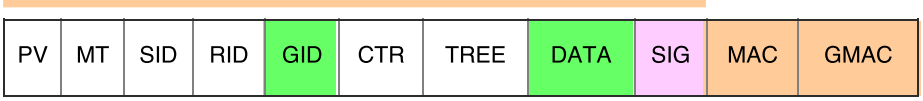


Figure 5.1.: The tree announcement message is used to declare the key tree to a new user. Compared to the previous key update message (figure 3.4) the configuration is encrypted together with all BK signatures (DATA). The group id (GID) and group MAC (GMAC) allows all other members to verify that another member announced the tree.

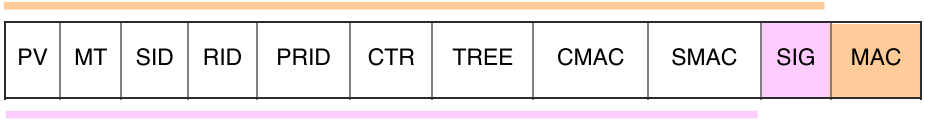


Figure 5.2.: The updated tree update message contains a MAC created with the new tree and two control codes to verify the configuration (CMAC) and member list (SMAC), because those two values were only encrypted and readable for the new user in the tree announcement message.

Through the changed message flow (figure 5.3) the approver sends the tree together with the list of long-lived DSA signatures of all leafs to the candidate. This is more robust against not responding members, because the new member is already reachable and willing to calculate the updated tree, because he wants to join the group. This approach does not only prevent deadlocks, but also complicates DOS attacks[1] by letting the attacker compute some required tasks. While the number of messages have remained constant, the overall security is not weakened compared to the original protocol. A malicious candidate, who is considered trustworthy, gets the same information as before, like the public key tree or the list of DSA signatures. Compared to the old tree update message, the new format hides the configuration from eavesdroppers by encrypting it with a secret only known by the sender and trusted candidate. (See section 4.2 for the analysis of information disclosure.) Figure 5.4 clearly shows that, with both tweaks, a single user (approver or sponsor) is no longer capable of blocking the entire group.

Besides the changes above, some minor enhancements were made:

> *To reduce the number of error messages, every member should delay the sending of this message by a random time between 0 and 3 seconds. If they receive another error message in this time, they should abort the transfer. (D.3.4.1)*

This addition was necessary to prevent flooding the broadcast channel with error messages in large groups, especially after removing the instruction that the last active member has to send an error message to inform the candidate about an existing group. This removal was required, because the last active user could be not reachable for a long time. Instead every user should send an error message with the proposed limitation.

> *Any message containing the string "?TBG-OTR Error:",* ***optional followed by the group fingerprint, plus the character ":"*** *is a TBG-OTR Error Message. (D.3.1.2)*

[1] An attacker could try to overcharge existing members with the calculation of multiple tree updates through a trusted DSA signed request message.

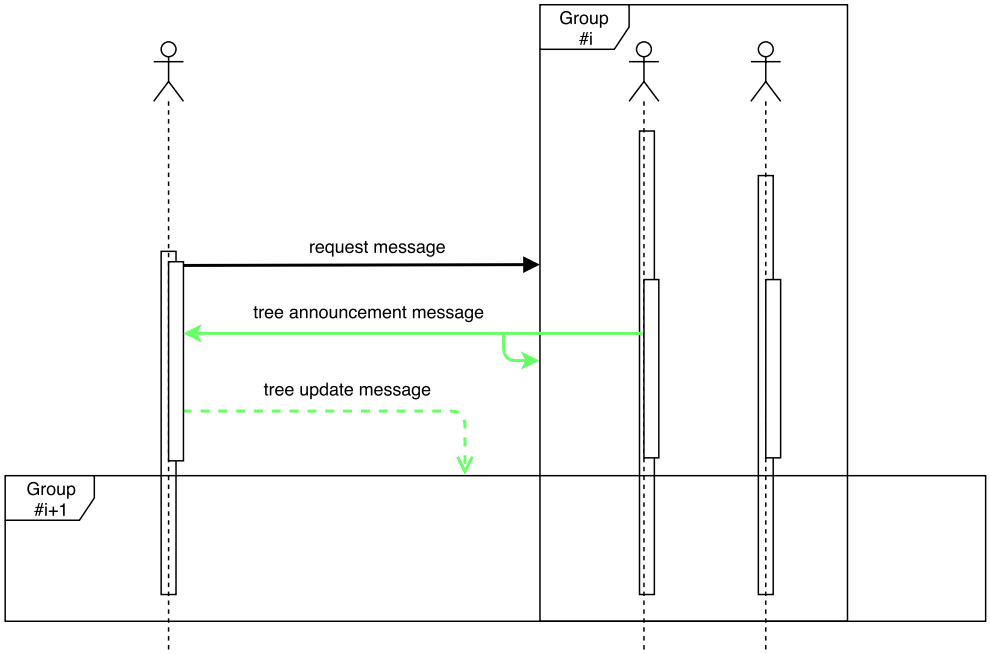


Figure 5.3.: This diagram shows the process from a join request to a tree update message with one required approval. In this case the tree announcement message functions as approval.

Multiple groups could coexist in one broadcast channel. Therefore it was unavoidable to add a reference to the source of the message. With this fingerprint a new user can now also address which group[2] he exactly wants to join (see figure 5.5).

> *Type 4 Public long-lived DSA key*
> *A freshly joined user has to send his public long-lived DSA key immediately after the join was completed (PUBKEY). (D.3.2.5)*

As mentioned in section 4.4, people start trusting other people after some time. To recognize them, they need the public long-lived DSA key to verify the signature of new join requests, which is now possible with the help of a new TLV record. The requirement to send this key immediately after a successful join is also used to reveal the short-lived DSA key, which reduces the time to intercept that identity.

> *Number of required confirmations (Default=1). Needs to be always 0 or greater.* ***If this number is equal or greater than the current number of members, a confirmation from all members is required.*** *(D.3.2.3)*

[2]Because the fingerprint does not reveal any information (e.g. participants, topic) about the group, the user has to make his decision with the help of other information channels (e.g. calls, plaintext messages).

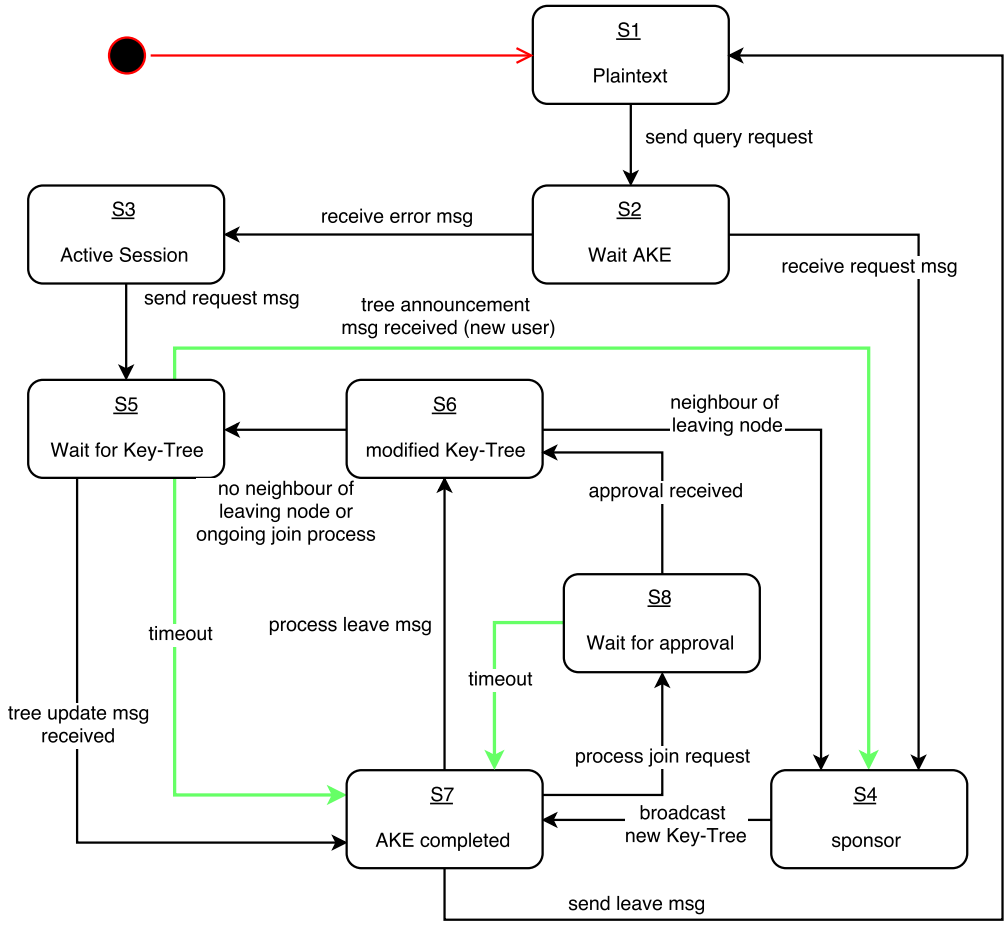


Figure 5.4.: The changes in the AKE led only to small changes in the state diagram. Now the new user appears as sponsor after he receives the tree announcement message and it is allowed to abort an ongoing join or leave process after an appropriate amount of time. If a member can send the last required approval, he instead sends the tree announcement message in transition S8-S6.

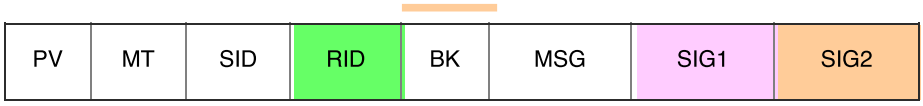


Figure 5.5.: Compared to the original structure (figure 3.3), the new request message has a receiver id (RID).

This annotation was important, because otherwise, if taken literally, no group can be created with a value greater than one. In particular while adding a new user, the first user would wait forever, because he is only able to provide one approval.

5.2. Verification

There are two kinds of verifications in the TBG-OTR protocol:

- Verification of the owner of a key.
- Verification of the group view (session parameters).

The first kind is what is usually meant by verification of the long-lived signing key, to ensure that a person is really who he is claiming to be. This is done by comparing the fingerprint of the used key either by meeting in person, phone calls or the Socialist Millionaires' Protocol. While this verification process always needs user actions, the second kind of verification can be done automated, because it verifies that other members have the same view about the group. Such an approach helps to detect MitM attacks as demonstrated in section 4.5 and ensures that short-lived signing keys are linked correctly to their long-lived relatives. This process has to ensure that the following requirements are met for a specific group (G) with a set of members (C):

1. The owner (O) of a long-lived signing key (L) is part of G.
2. O is using the short-lived signing key S.
3. O assumes C.
4. O is using the key tree T.
5. The verification is deniable.

To verify that those properties hold, the new version of TBG-OTR uses a request-response approach. If a member[3] wants to know that another person has the same view about the group, he creates a two-party group with the desired member by creating a root key from booth DH keys. The advantage of this mini group is, that all message types can be used as usual and no extra key exchange is needed. This also means that all data is signed with the short-lived signing key and authenticated with a MAC key only known to both parties. Both messages contain only the HMAC generated with the group MAC key of the group id, member list and both signing keys of the receiver, to prevent revealing sensible information to a pseudo user and to

[3]Or the chat client of that member.

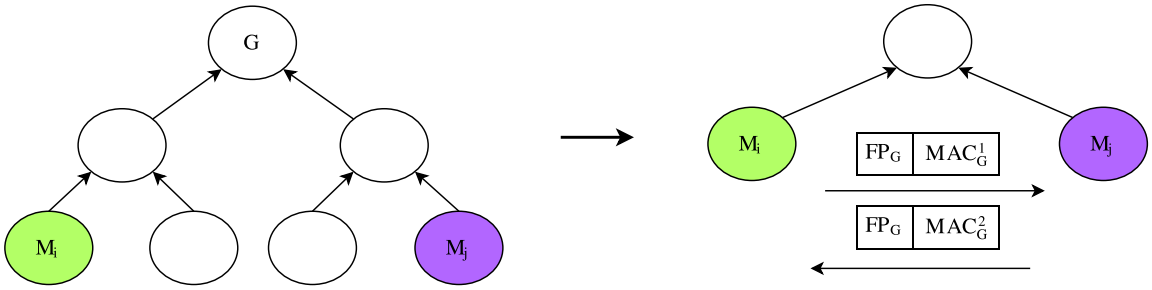


Figure 5.6.: If M_i wants to verify his view about the group (G) with M_j, he creates a two-party group from both DH keys and sends a normal data message (figure 5.1) with a type 6 TLV. This record contains the group id and a MAC produced with the group secret of G.

prove group membership (1) with the same key tree (4). As long as this information is also signed by the short-lived key and transfered in a channel which is only accessible by the owner of the long-lived key, requirement (2) is satisfied. All exchanged data is still deniable (5), because, similar to OTR, nobody in that channel can prove[4] that a message was send by the other party. To complete the verification every user has to confirm a request, if the MAC was correct, with a message containing the response MAC. If an error occurs, or the user is not willing to confirm, he has to send an abort message. This process is visualized in figure 5.6.

> *The number of pairwise group parameter verifications, depends on the level of confidentiality. (D.3.4.3)*

Every user can decide on his own, how many verifications he needs to trust a group. This number depends especially on the discussed topics. While for a medium confidential topic it would be enough to verify members at strategic[5] nodes in the tree, top-secret discussions would require to check all members. This liberty is needed for the protocol to be still robust against not responding members and does not influence the security at all.

> *No established member should initiate a group parameter verification with a new member. (D.3.4.3)*

Established members do not need to verify group parameters with a new member, because they can comprehend the evolution of the key tree on the basis of the signed join request. Even if they are not able to verify the correctness of all parts of the

[4]Both parties are able to generate the same MAC and know only that the other party send a message, because they did not send it by themself. Also the short-lived key could be owned by any party and therefore this signature does not influence deniablity either.

[5]A strategic node could be a node which is able to verify large parts of the tree.

tree, they have to trust that a member who could, would protest against that unauthorized change. The above requirement also prevents new members to get flooded by verification requests.

5.3. Miscellaneous

As mentioned in section 4.5, SHA-1 is no longer collision resistant and should be replaced in the future. For this reason we updated the hash algorithm, used to compute fingerprints and keys, to SHA-256. To reduce packet length only the first 20 bytes of the hash are used as fingerprint. This increases the possibility of collisions, but is still collision resistant.

> *If the sponsor, who is required to finish a leave request, is not responding, every remaining member should mark the leaving member and wait until the sponsor responses again. (D.3.4.2)*

In order to remove a member from the key tree it is necessary to move his neighbour to a new position and recalculate all keys up to the root. The most efficient way is to move the neighbour one level up and let him calculate all desired intermediate results. The other possibility would be to remove both members and rejoin the neighbour. Depending on the used approach the key tree could become unbalanced or multiple responding members were required. For this reason the new protocol version handles a leave message as a notification, which can be processed if possible. This does not influence the security, because all existing members know that the leaving member can still read all further messages.

> *If a sponsor left a group, before he updated the tree, he should be marked and all members should check if they are the sponsor for both gone members. (D.3.4.2)*

To prevent a lot of dead nodes in a tree, it is necessary to handle the case if both neighbours left a group. In such a case one of the members of the neighbour sub-tree has to remove the complete two-party tree and level their own sub-tree up. (See figure 5.7 for a visualisation.)

> *During an ongoing join process, nobody should leave a group. (D.3.4.2)*

If a member leaves a group during the tree announcement and update message, he will only be removed from the old group (figure 5.8), because otherwise it is unclear for the new user why this member was removed.

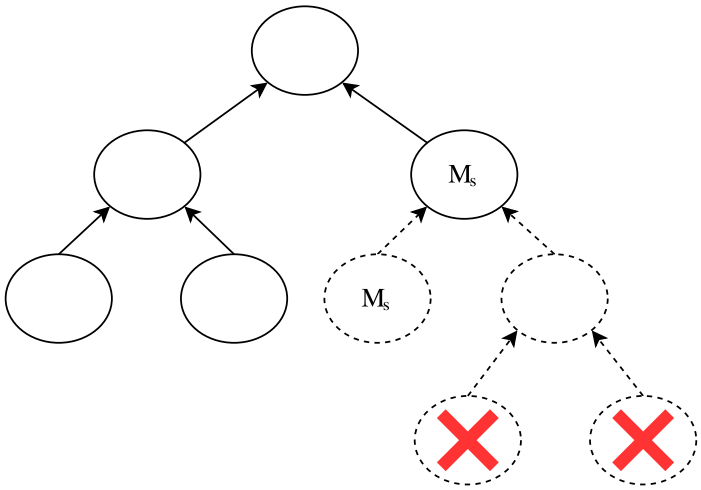


Figure 5.7.: After two neighbours left the group the sponsor of their parent node has to remove them both and update the tree accordantly.

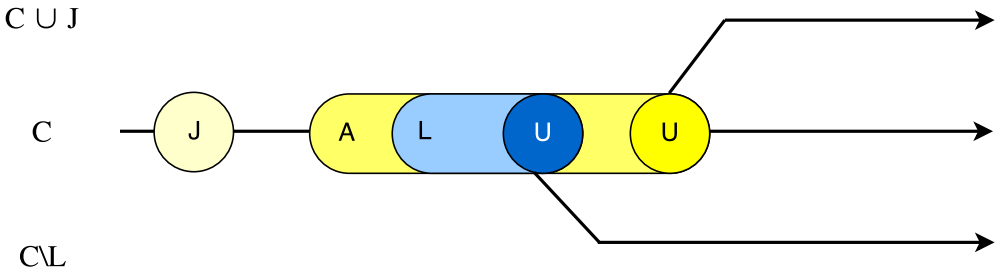


Figure 5.8.: After a join request (J) is accepted, the tree is announced (A) and updated (U) by the new user. If a member leaves (L) the group between these two messages, two different groups will derive, because the new user does not know about the leave notification.

6. Future work & Conclusion

6.1. Future work

In the previous chapter the current protocol was further improved by a more robust join process, new verification possibilities, and some minor changes. As a next step, this improved protocol has to be theoretically analysed and practically tested with different groups and users. It should be examined how large the resulting packets are and if the size of the tree update message can be reduced by announcing only the changed parts of the tree. In practical tests it will also show if the delay durations are chosen correctly, or if a measurement of the round-trip time can be useful. In further work it should be examined if there are large ephemeral groups which would benefit from a divide-and-conquer join and how this process could remain robust against not responding existing members. The verification process could also be further improved by investigating how many verifications have to be done and which strategies in selecting verification nodes are most effective. Even the use of a concept similar to a web-of-trust, in which a verified user shares his verifications, should be considered and tested. In the analysis multiple issues arise regarding the leave process and show that a more robust leave strategy is needed. It should be examined how this could be more robust and consistent through the use of protests to remove members also from derived groups. Finally, possibilities of multi-client support should be considered to cover common use cases and in even further work, different user interfaces could be developed and compared to improve user experience with conversation branches, derived groups and number of verifications.

6.2. Conclusion

In this master thesis we proposed the first secure multi-party communication protocol which is robust against not responding members. For this we analysed behaviours of groups, like the admission process or course of conversation, and derived from this properties for the new protocol. We showed that not all present members pay always attention and for this reason relations between messages are more important than consensus about all transferred messages. Our observations about the admission process with transitive trust allowed us to build a key exchange based on a Diffie-Hellman tree which does not require all members to be online at the same time. The limited number of message origins and the fact that a group which decided to cheat

us as a whole, is always capable of doing so, allowed us to use ephemeral signing keys, which were trusted on first use if nobody did protest, to authenticate messages. However, the analysis of the first protocol version showed that multiple attacks were possible. In the proposed next revision, timeouts were eliminated through clearer instructions and a new join process, which also prevents an UKS attack on the key exchange. With an integrated verification process, to verify group parameters, we complicate MitM attacks even for trusted users.

The robustness against not responding members could not exclusively be used for online communication, but also for other applications which could benefit from the tree structure like file sharing. TBG-OTR meets all our requirements by using human behaviour and it will be interesting how this concept will develop in the future.

A. Bibliography

[1] Chris Alexander and Ian Goldberg. "Improved User Authentication in Off-the-record Messaging". In: *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*. WPES '07. New York, NY, USA: ACM, 2007, pp. 41–47. ISBN: 978-1-59593-883-1. DOI: 10.1145/1314333.1314340. URL: http://dl.acm.org/citation.cfm?id=1314340.

[2] Jim Alves-foss. "An Efficient Secure Authenticated Group Key Exchange Algorithm for Large and Dynamic Groups". In: *proc. 23 RD national information systems security conference*. 2000, pp. 254–266.

[3] D. Augot et al. "An Efficient Group Key Agreement Protocol for Ad Hoc Networks". In: *Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks* (), pp. 576–580. DOI: 10.1109/WOWMOM.2005.26. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1443570.

[4] Mihir Bellare. "New Proofs for NMAC and HMAC: Security Without Collision-Resistance". In: *Advances in Cryptology - CRYPTO 2006: 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 602–619. ISBN: 978-3-540-37433-6. DOI: 10.1007/11818175_36. URL: http://dx.doi.org/10.1007/11818175_36.

[5] S. M. Bellovin and M. Merritt. "Limitations of the Kerberos Authentication System". In: *SIGCOMM Comput. Commun. Rev.* 20.5 (Oct. 1990), pp. 119–132. ISSN: 0146-4833. DOI: 10.1145/381906.381946. URL: http://doi.acm.org/10.1145/381906.381946.

[6] Philip A Bernstein, Nathan Goodman, and Vassos Hadzilacos. "Concurrency Control and Recovery in Database Systems". In: *ACM Transactions on Database Systems* (1987), p. 370. DOI: 10.1145/1994.2207.

[7] J Bian, R Seker, and U Topaloglu. "Off-the-Record Instant Messaging for Group Conversation". In: *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*. 2007, pp. 79–84. DOI: 10.1109/IRI.2007.4296601.

[8] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. "Distinguisher and Related-Key Attack on the Full AES-256 (Extended Version)". In: *Cryptology ePrint Archive* 241 (2009). URL: http://eprint.iacr.org/2009/241.

[9] Alex Biryukov et al. *Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds*. Cryptology ePrint Archive, Report 2009/374. 2009.

[10] Simon Blake-Wilson and Alfred Menezes. "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol". In: *Public Key Cryptography: Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99 Kamakura, Japan, March 1–3, 1999 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 154–170. ISBN: 978-3-540-49162-0. DOI: 10.1007/3-540-49162-7_12. URL: http://dx.doi.org/10.1007/3-540-49162-7_12.

[11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. "Biclique cryptanalysis of the full AES". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7073 LNCS. Springer Berlin Heidelberg, 2011, pp. 344–371. ISBN: 9783642253843. DOI: 10.1007/978-3-642-25385-0_19. URL: http://link.springer.com/10.1007/978-3-642-25385-0_19.

[12] Timo Brecher, Emmanuel Bresson, and Mark Manulis. "Fully robust tree-diffie-hellman group key exchange". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5888 LNCS (2009), pp. 478–497. ISSN: 03029743. DOI: 10.1007/978-3-642-10433-6_33.

[13] Emmanuel Bresson and Dario Catalano. "Constant Round Authenticated Group Key Agreement via Distributed Computation". In: 04 (2004), pp. 115–127. ISSN: 03029743. DOI: 10.1007/978-3-540-24632-9_9.

[14] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. "Provably Authenticated Group Diffie-Hellman Key Exchange – The Dynamic Case [Full version]". In: *Advances* 01 (2001), pp. 290–309.

[15] Scott Chacon and Ben Straub. *Pro Git*. 2nd. Berkely, CA, USA: Apress, 2014. ISBN: 1484200772, 9781484200773. URL: https://git-scm.com/book/en/v2.

[16] Cryptocat. *Multiparty Protocol Specification*. visited on 03-24-2016. URL: https://github.com/cryptocat/cryptocat/wiki/Multiparty-Protocol-Specification.

[17] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. "Secure off-the-record messaging". In: *Proceedings of the 2005 ACM workshop on Privacy in the electronic society - WPES '05* (2005), p. 81. DOI: 10.1145/1102199.1102216. URL: http://portal.acm.org/citation.cfm?doid=1102199.1102216.

[18] Whitfield Diffie and Martin E Hellman. "New Directions in Cryptography". In: *IEEE TRANSACTIONS ON INFORMATION THEORY* 6 (1976).

[19] Niels Ferguson et al. “Improved Cryptanalysis of Rijndael”. In: *Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 213–230. ISBN: 978-3-540-44706-1. DOI: 10.1007/3-540-44706-7_15. URL: http://dx.doi.org/10.1007/3-540-44706-7_15.

[20] N Fips. “197: Announcing the advanced encryption standard (AES)”. In: *... Technology Laboratory, National Institute of Standards ...* 2009 (2001), pp. 8–12. ISSN: 13534858. DOI: 10.1016/S1353-4858(10)70006-4. URL: csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[21] Sally Floyd et al. “A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing”. In: *IEEE/ACM Transactions on Networking* (1997).

[22] Ian Goldberg et al. “Multi-party Off-the-Record Messaging”. In: *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09* (2009), p. 358. ISSN: 15437221. DOI: 10.1145/1653662.1653705. URL: http://dl.acm.org/citation.cfm?id=1653662.1653705.

[23] Jonathan Katz and Moti Yung. “Scalable protocols for authenticated group key exchange”. In: *Journal of Cryptology* 20.1 (2007), pp. 85–113. ISSN: 09332790. DOI: 10.1007/s00145-006-0361-5.

[24] Yongdae Kim, Adrian Perrig, and Gene Tsudik. “Simple and Fault-tolerant Key Agreement for Dynamic Collaborative Groups”. In: *Proceedings of the 7th ACM Conference on Computer and Communications Security*. CCS '00. New York, NY, USA: ACM, 2000, pp. 235–244. ISBN: 1-58113-203-4. DOI: 10.1145/352600.352638. URL: http://doi.acm.org/10.1145/352600.352638.

[25] Yongdae Kim, Adrian Perrig, and Gene Tsudik. “Tree-based Group Key Agreement”. In: *ACM Trans. Inf. Syst. Secur.* 7.1 (2004), pp. 60–96. ISSN: 1094-9224. DOI: 10.1145/984334.984337. URL: http://eprint.iacr.org/2002/009.pdfhttp://doi.acm.org/10.1145/984334.984337.

[26] T Kivinen and M Kojo. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*. RFC 3526 (Proposed Standard). 2003. URL: http://www.ietf.org/rfc/rfc3526.txt.

[27] Hugo Krawczyk. “SIGMA: The ‘SIGn-and-MAc’ Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols”. In: *Advances in Cryptology - CRYPTO 2003* 2729 (2003), pp. 400–425. ISSN: 03029743. DOI: 10.1007/978-3-540-45146-4_24. URL: http://www.ee.technion.ac.il/~hugo/sigma.htmlhttp://dx.doi.org/10.1007/978-3-540-45146-4_24.

[28] Leslie Lamport. “The Part-Time Parliament”. In: *ACM Transactions on Computer Sys-tems* 16.2 (1998), pp. 133–169. ISSN: 07342071. DOI: 10.1145/279227.279229.

[29] Hong Liu, Eugene Y Vasserman, and Nicholas Hopper. "Improved Group Off-the-record Messaging". In: *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*. WPES '13. New York, NY, USA: ACM, 2013, pp. 249–254. ISBN: 978-1-4503-2485-4. DOI: 10.1145/2517840.2517867. URL: http://doi.acm.org/10.1145/2517840.2517867.

[30] National Institute of Standards and Technology. "Digital Signature Standard (DSS)". In: *FIPS PUB 186-4* July (2009), pp. 1–119. DOI: 10.6028/NIST.FIPS.186-4. URL: http://dx.doi.org/10.6028/NIST.FIPS.186-4.

[31] Open Whisper Systems. *Private Group Messaging*. visited on 02-28-2016. URL: https://whispersystems.org/blog/private-groups/.

[32] R L Rivest, A Shamir, and L Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126. ISSN: 00010782. DOI: 10.1145/359340.359342. arXiv: arXiv:1011.1669v3.

[33] Robert Shirey. *Internet Security Glossary (RFC2828)*. 2000. URL: https://www.ietf.org/rfc/rfc2828.txt.

[34] Dale Skeen and Michael Stonebraker. "A Formal Model of Crash Recovery in a Distributed System". In: *IEEE Transactions on Software Engineering* SE-9.3 (1983), pp. 219–228. ISSN: 00985589. DOI: 10.1109/TSE.1983.236608. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1703048.

[35] M Steiner, G Tsudik, and M Waidner. "CLIQUES: a new approach to group key agreement". In: *Proceedings 18th International Conference on Distributed Computing Systems Cat No98CB36183* 2984 (1998), pp. 380–387. ISSN: 1063-6927. DOI: 10.1109/ICDCS.1998.679745. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=679745.

[36] Marc Stevens, Pierre Karpman, and Thomas Peyrin. "Freestart collision for full SHA-1". In: *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 459–483. ISBN: 978-3-662-49890-3. DOI: 10.1007/978-3-662-49890-3_18. URL: http://dx.doi.org/10.1007/978-3-662-49890-3_18.

[37] Qianhong Wu et al. "Asymmetric Group Key Agreement". In: *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques* 5479 (2009), pp. 153–170. DOI: 10.1007/978-3-642-01001-9_9.

[38] Paul D Yacoumis. “On the Security of the Advanced Encryption Standard”. 2005. URL: http://diamond.boisestate.edu/~liljanab/MATH509/AES-security.pdf.

B. List of abbreviations

AES Advanced Encryption Standard

MAC Message Authentication Code

HMAC Keyed-Hash Message Authentication Code

DH Diffie-Hellman

TBG-OTR Tree-based Group Off-the-record

UKS Unkown Key-Share

MitM Man-in-the-Middle

DOS Denial of Service

SIGMA SIGn-and-MAc

D&C divide and conquer

GKE Group Key Exchange

AGKE Authenticated Group Key Exchange

AES Advanced Encryption Standard

DSA Digital Signature Algorithm

SMP Socialist Millionaires' Protocol

FPD fundamental problem of deniability

TLV Type-Length-Value

PFS Perfect Forward Secrecy

C. TBG-OTR version 0.1

This protocol is heavily influenced by OTR v3 from cypherpunks and uses the Tree-based Group Key Agreement protocol from Y. Kim, A. Perrig and G. Tsudik in an authenticated manner.

Author: Klaus Herberth

C.1. Very high level overview

TBG-OTR assumes a broadcast channel which delivers messages in the same order to all participants.

1. A new member sends a request to join.
2. After a group approval all existing members, which agree with the decision, add the new member to the group.
3. All group members exchange data messages to send information.

C.2. High level overview

C.2.1. Joining

This section outlines Tree-based Group Key Agreement used as AKE. All exponentiations are done modulo a particular 1536-bit prime, and g is a generator of that group, as indicated in the detailed description below. Public DH keys, also called blinded keys, are bkeys or bk and calculated as g^{key}.

1. New member M_{n+1} broadcasts request (signed with long-live DSA key) for join to current Group C
2. $M_i \in C$ knows \$ M_{n+1} \$ or gets to know him. If he thinks this member is trustful he sends an approval to C. (Multiple approvals could be required)
3. Every member
   - updates his binary key tree by adding new member node and new intermediate node.
   - removes all keys and bkeys from the leaf node related to the sponsor to the root node.
4. The sponsor M_s additionally

- generates new share and computes all [key, bkey] pairs on the key-path.
- broadcasts updated tree including only bkeys.

5. Every member computes group key using the updated tree.

Now M_{n+1} knows who is aware of this key. To verify those he has to contact everybody, because pseudo members could be possible.

C.2.2. Exchanging Data

This section outlines the method used to protect data being exchanged between all members. As above, all exponentiations are done modulo a particular 1536-bit prime, and g is a generator of that group, as indicated in the detailed description below.

Suppose M_i has a message to send to all group members (C): - M_i: 1. Uses the key tree to generate a shared secret (s) for C, and generates the sending AES key (ek) and the sending MAC key (mk), as detailed below. 2. Picks a value of the counter, ctr, so that the tuple (s, ctr) is never the same for more than one data message. 3. Picks last received or referenced message id as history. 4. Generates signature (sig) of history + msg with short-lived DSA key. 5. Computes T = (sender id, receiver id, ctr, AES-CTR$_{ek,ctr}$(history + msg + sig)) 6. Sends T, MAC_{mk}(T) - Every $M_m \in C$: 1. Uses key tree and receiver id to generate a shared secret. 2. Uses sender id to compute receiving AES key (ek) and receiving MAC key (mk), as detailed below. 3. Uses mk to verify MAC_{mk}(T) 4. Uses ek and ctr to decrypt AES-CTR$_{ek,ctr}$(history + msg + sig) 5. Verifies signature with known DSA key

C.2.3. Leaving

After a member (M_d) is gone, the following steps have to be processed:

- Every member:
  1. updates key tree by removing the leaving member node and relevant parent node.
  2. removes all keys and blinded keys from the leaf node related to the sponsor to the root node.
- Sponsor M_s additionally:
  1. generates new share and computes all [key, bkey] pairs on the key-path.
  2. broadcasts updated tree including only bkeys
- Every member computes the group key using the updated key tree.

C.3. Details of the protocol

C.3.1. Unencoded messages

This section describes the messages in the TBG-OTR protocol that are not base-64 encoded binary.

C.3.1.1. TBG-OTR Query Messages

If a user (A) wants to start a TBG-OTR session he sends a message starting with ?TBG-OTR? optionally followed by an explanation. Two cases are possible:

1. There is already an active TBG-OTR session in the broadcast channel. The last active member of that group should send a TBG-OTR error messages, followed by a request message from user A to join that group.
2. There is no active TBG-OTR session in the broadcast channel. All present users who want to join should send a request message and user A acts as first sponsor for the first received request message. All other messages are handled as described below.

C.3.1.2. TBG-OTR Error Messages

Any message containing the string "?TBG-OTR Error:" is an TBG-OTR Error Message. The following part of the message should contain human-readable details of the error.

C.3.2. Encoded messages

This section describes the byte-level format of the base-64 encoded binary TBG-OTR messages. The binary form of each of the messages is described below. To transmit one of these messages, construct the ASCII string consisting of the nine bytes "?TBG-OTR:", followed by the base-64 encoding of the binary form of the message, followed by the byte ".".

For the Diffie-Hellman group computations, the group is the one defined in RFC 3526 with 1536-bit modulus (hex, big-endian):

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

and a generator (g) of 2. Note that this means that whenever you see a Diffie-Hellman exponentiation in this document, it always means that the exponentiation is done modulo the above 1536-bit number.

C.3.2.1. Data types

Bytes (BYTE)
: 1 byte unsigned value

Shorts (SHORT)
: 2 byte unsigned value, big-endian

Ints (INT)
: 4 byte unsigned value, big-endian

Multi-precision integers (MPI)
: 4 byte unsigned len, big-endian len byte unsigned value, big-endian (MPIs must use the minimum-length encoding; i.e. no leading 0x00 bytes. This is important when calculating public key fingerprints.)

Opaque variable-length data (DATA)
: 4 byte unsigned len, big-endian len byte data

Initial CTR-mode counter value (CTR)
: 8 bytes data

Message Authentication Code (MAC)
: 20 bytes MAC data

C.3.2.1.1. Public keys and signatures

TBG-OTR users have long-lived public DSA keys that they use for authentication (but not encryption). And short-lived public DSA keys for message authentication. Both should be never be linked and the short-lived key should only be verified over a deniable channel (e.g. OTR session). The current version of the TBG-OTR protocol only supports DSA public keys, but there is a key type marker for future extensibility.

OTR public DSA key (PUBKEY)
: Pubkey consist of the type and the key parameters Pubkey type (SHORT)

  - DSA public keys have type 0x0000

  DSA public key parameters

  - p (MPI)
  - q (MPI)
  - g (MPI)
  - y (MPI)

TBG-OTR public DSA keys are used to generate signatures; different types of keys produce signatures in different formats. The format for a signature made by a DSA public key is as follows:

DSA signature (SIG)
(len is the length of the DSA public parameter q, which in current implementations must be 20 bytes, or 160 bits)
len byte unsigned r, big-endian
len byte unsigned s, big-endian

C.3.2.1.2. Fingerprints

TBG-OTR public keys (DSA and DH) have fingerprints, which are hex strings that serve as identifiers for the public key. The fingerprint is calculated by taking the SHA-1 hash of the byte-level representation of the public key.

Public key fingerprint (FP)
20 bytes SHA1 data

C.3.2.1.3. Type-Length-Value record

Each TLV record is of the form:

Type (SHORT)
The type of this record. Records with unrecognized types should be ignored.
Length (SHORT)
The length of the following field
Value (len BYTEs)
(where len is the value of the Length field) Any pertinent data for the record type.

Some TLV examples:

`\x00\x01\x00\x00`
A TLV of type 1, containing no data

`\x00\x00\x00\x05\x68\x65\x6c\x6c\x6f`
A TLV of type 0 (`\x00\x00`) with length 5 (`\x00\x05`), containing the value "hello" (`\x68\x65\x6c\x6c\x6f`).

C.3.2.2. Request Message

This is the first message of the AKE. Alice sends it to the group to request acceptance.

C.3.2.2.1. Format

Protocol version (SHORT)
The version number of this protocol is 0x0001.

Message type (BYTE)
The request message has type 0x01.
Sender id (FP)
The long-lived DSA fingerprint of the person requesting acceptance.
Blinded key (DATA)
The public DH key of the person requesting acceptance.
Plaintext message (DATA)
An optional plaintext message which is displayed to all members.
Signature (SIG)
The DSA signature, using the long-lived DSA key, of everything from the protocol version to the plaintext message.
Signature (SIG)
The DSA signature, using the long-lived DSA key, of the blinded key. This signature is transfered along with the updated key tree.

C.3.2.3. Tree Message

The sponsor sends this message after a group modification (join, leave, timeout).

C.3.2.3.1. Format

Protocol version (SHORT)
The version number of this protocol is 0x0001.
Message type (BYTE)
The tree message has type 0x02.
Sender id (FP)
The public DH key fingerprint of the sponsor.
Receiver id (FP)
The public DH fingerprint of the new derived group key.
Previous receiver id (FP)
The public DH fingerprint of the group key which triggered this tree update.
Configuration (DATA)
Initial configuration of the group. If no configuration is set (0x00 00 00 00) the default values apply. Currently it is not allowed to change this value after group initialisation. And if it does, it should trigger a protest. The currently defined TLV (type-length-value) options are:

- Type 0: Number of required confirmations (Default=1). Needs to be always 0 or greater.
- Type 1: Number of seconds after a sponsor should be considered as offline (Default=8).

Top half of counter init (CTR)
: This should monotonically increase (as a big-endian value) for each message sent with the same (sender id, recipient id) pair, and must not be all 0x00.

Serialized tree with BK (DATA)
: Serialize the new key tree in preorder with every BK (DATA) followed by either 0x01 (BYTE) for leafes or 0x00 for empty nodes.

Encrypted BK signatures (DATA)
: Order all long-lived DSA leaf signatures (SIG) from left to right followed by there corresponding key id, encrypt it using AES128-CTR and encode it as DATA field. The initial counter is a 16-byte value whose first 8 bytes are the above "top half of counter init" value, and whose last 8 bytes are all 0x00.

Signature (SIG)
: The DSA signature, using the short-lived DSA key, of everything from the protocol version to the encrypted signatures.

Authenticator (MAC)
: The SHA1-HMAC, using the new MAC key, of everything from the protocol version to the encrypted message.

C.3.2.4. Data Message

This message is used to transmit a private message to the correspondent, and consists of:

- MAC of the referenced or last received message (MAC).
- Plaintext message consists of a human-readable message (encoded in UTF-8, optionally with HTML markup) (DATA).
- Zero or more TLV (type/length/value) records (with no padding between them).
- DSA signature with short-lived key of the first 3 items.

The currently defined TLV records for data messages are:

Type 0 Padding
: The value may be an arbitrary amount of data, which should be ignored. This type can be used to disguise the length of the plaintext message.

Type 1 Disconnected
: If the user requests to close the private connection, you may send a message (possibly with empty human-readable part) containing a record with this TLV type just before you discard the session keys, and transition to MSGSTATE_PLAINTEXT (see below). If you receive a TLV record of this type, you should inform the user that this member has closed his end of the private connection.

Type 2 Confirmation
: This type represents the confirmation of a join request and starts the tree update if the number of confirmations is greater or equal to the configured value.

Type 3 Public DSA key
: If a user sends a data message the first time after a new user joined, he has to send his public short-lived DSA key (PUBKEY).

Type 4 protest
: If a fraud attempt is detected, every client should protest. If a protest message is received and believable, the affected message should be discard and possible key updates revoked. The value of a TLV type 4 consists of a reason (SHORT) followed by the affected message id (MAC). Reasons for a protest are:

  - 0x00 Other
  - 0x01 Tree manipulated
  - 0x02 Configuration changed unauthorized
  - 0x03 Member uses multiple short-lived DSA keys

C.3.2.4.1. Format

Protocol version (SHORT)
: The version number of this protocol is 0x0001.

Message type (BYTE)
: The data message has type 0x03.

Sender id (FP)
: The public DH key fingerprint of the sponsor.

Receiver id (FP)
: The public DH fingerprint of the group key.

Top half of counter init (CTR)
: This should monotonically increase (as a big-endian value) for each message sent with the same (sender id, recipient id) pair, and must not be all 0x00.

Encrypted message (DATA)
: Using the corresponding group key (receiver id), perform AES128 counter-mode encryption of the message.

Authenticator (MAC)
: The SHA1-HMAC, using the group MAC key, of everything from the protocol version to the encrypted message.

C.3.3. Key Management

For each group a client should keep track of:

- At least 2 recent BK-Trees, until every member sends at least one message with the new key.
- Signatures of all public DH leaf keys.
- A mapping between each members used DH (leaf) and short-lived DSA key

When starting a private conversation with a group, generate one DH key pair for yourself. Note that all DH keys should have a private part that is at least 320 bits long.

When you send a data message
: Use the root DH key of the desired group to calculate session keys, as outlined below. Use the "sending AES key" to encrypt message and the "sending MAC key" to calculate its authenticator.

When you receive a data message
: Use the ids in the message to calculate session keys, as outlined below. Use the "receiving MAC key" to verify the authenticator. If sender id is no leaf in the tree, the receiver is is no node or the MAC does not match, reject the message.

C.3.3.1. The blinded key tree

TBG-OTR uses Tree-based Group Key Agreement to calculate shared secrets:

- All members are leafes in a binary tree.
- Every node contains a blinded key (g^{s_i}) of a secret (s_i).
- All members need to share the same tree in order to calculate the same secret.
- To calculate the root secret, at least one private key is needed. At the beginning this is the private part of a leaf.
- With the known secret nearest to the root ($s_{i,0}$) and the public key of the neighbor node ($g^{s_{i,1}}$) the private key $s_{i-1,0}$ of the parent node is calculated with $g^{s_{i,0}s_{i,1}}$.
- Repeat the last step until you get the root secret (s).

C.3.3.1.1. Adding a member

To keep the tree compact, a new node (and therefore member) should be added to the rightmost shallowest leaf node (s_i). The member which owns the private part of node s_i is called sponsor and has to perform the following actions:

1. Delete all blinded keys on the way up to the root ($s_{i..0}$).
2. Add the blinded key of his own node and from the new member as children to s_i.
3. Calculate all blinded keys up to the root, as described in the previous section.
4. Broadcast the updated tree with all blinded keys.

C.3.3.1.2. Removing a member

If a member (m) has to be removed, the neighbor of him is called the sponsor (s_i) and has to perform the following actions:

1. Delete all blinded keys on the way up to the root ($s_{i..0}$).

2. Replace the parent node with his node and remove the gone member m.
3. Calculate all blinded keys up to the root, as described in the section above.
4. Broadcast the updated tree with all blinded keys.

C.3.3.2. Computing AES and MAC keys

Based on the root secret s each member computes 2 values:

- One 128-bit AES encryption key code
- One 160-bit SHA1-HMAC key m

These keys are calculated as follows:

- Write the value of s as a minimum-length MPI, as specified above. Let this (4 + len)-byte value be "secbytes".
- For a given salt, define h1(salt) to be the 160-bit output of the SHA-1 hash of the salt followed by secbytes.
- The "sending AES key" is the first 16 bytes of h1(own public DH key id).
- The "sending MAC key" is the 20-byte SHA-1 hash of the 16-byte sending AES key.
- For every other member the "receiving AES key" is the first 16 bytes of h1(their public DH key id).
- For every other member the "receiving MAC key" is the 20-byte SHA-1 hash of the 16 byte receiving AES key.

C.3.4. Fragmentation

Some networks may have a maximum message size that is too small to contain an encoded TBG-OTR message. In that event, the sender may choose to split the message into a number of fragments. This section describes the format of the fragments. All TBG-OTR clients must be able to assemble received fragments, but performing fragmentation on outgoing messages is optional.

C.3.4.1. Transmitting Fragments

If you have information about the maximum size of message you are able to send (the different IM networks have different limits), you can fragment an encoded TBG-OTR message as follows:

- Start with the TBG-OTR message as you would normally transmit it. For example, a Data Message would start with "?TBG-OTR:AAED" and end with ".".
- Break it up into sufficiently small pieces. Let the number of pieces be (n), and the pieces be piece[1],piece[2],...,piece[n].

- Transmit (n) fragmented messages with the following (printf-like) structure (as k runs from 1 to n inclusive): ?TBG-OTR|%x|%x,%hu,%hu,%s," , sender id, receiver id, k , n , piece[k]
- Note that k and n are SHORT, and each has a maximum value of 65535. Also, each piece[k] must be non-empty.

C.3.4.2. Receiving Fragments:

If you receive a message containing "?TBG-OTR|" (note that you'll need to check for this *before* checking for any of the other "?TBG-OTR:" markers):

- Parse it as the printf statement above into k, n, and piece.
- If the recipient's id does not match any node, and the listed receiver id is not zero, the recipient should discard the message and optionally pass along a warning for the user.
- Let (K,N) be your currently stored fragment number, and F be your currently stored fragment. (If you have no currently stored fragment, then $K = N = 0$ and $F =$ "".)
- If $k == 0$ or $n == 0$ or $k > n$, discard this (illegal) fragment.
- If $k == 1$:
  - Forget any stored fragment you may have
  - Store (piece) as F.
  - Store (k,n) as (K,N).
- If $n == N$ and $k == K + 1$:
  - Append (piece) to F.
  - Store (k,n) as (K,N).
- Otherwise:
  - Forget any stored fragment you may have
  - Store "" as F.
  - Store (0,0) as (K,N).

After this, if $N > 0$ and $K == N$, treat F as the received message.

C.3.5. The protocol state machine

A TBG-OTR client maintains separate state for every group. This state consists of two main state variables (message and authentication) and some other information. Additionally a client has to maintain keys, verification state a.s.o for every group member.

C.3.5.1. Message state

The message state variable (msgstate) controls what happens to outgoing messages typed by the user.

It can take the following values:

MSGSTATE_PLAINTEXT
: This state is the initial state and the only way back to it is through a explicit user request (UI operation), to prevent revealing information by mistake.

MSGSTATE_ENCRYPTED
: This state indicates that all outgoing messages are encrypted and is reached after successfull authentication.

C.3.5.2. Authentication state

The authentication state variable (authstate) indicates the process of the membership negotiation. It can take the following values:

AUTHSTATE_NONE
: This is the initial state and indicates that no authentication is in progress. It is also the only state in which a new authentication progress is allowed to start. If multiple requests arrive they should be handled one after another.

AUTHSTATE_AWAITING_CONFIRMATION
: After a new member sends the join request all current members enter this state.

AUTHSTATE_AWAITING_TREE
: After the required confirmations were send all current members enter this state. The new member enters this state right after he sends his request.

After the tree update was received:

- All members enter the state AUTHSTATE_NONE.
- The new member sets his msgstate variable to MSGSTATE_ENCRYPTED.
- The next join request is processed.

D. TBG-OTR version 0.2 (Diff)

This protocol is heavily influenced by OTR v3 from cypherpunks and uses the Tree-based Group Key Agreement protocol from Y. Kim, A. Perrig and G. Tsudik in an authenticated manner.

D.1. Very high level overview

TBG-OTR assumes a broadcast channel which delivers messages in the same order to all participants.

1. A new member sends a request to join.
2. After a group approval all existing members, which agree with the decision, add the new member to the group.
3. All group members exchange data messages to send information.

D.2. High level overview

D.2.1. Joining

This section outlines Tree-based Group Key Agreement used as AKE. All exponentiations are done modulo a particular 1536-bit prime, and g is a generator of that group, as indicated in the detailed description below. Public DH keys, also called blinded keys, are bkeys or bk and calculated as ~~g^{key}~~ $g^{key} \mod p$.

1. New member M_{n+1} broadcasts request (signed with long-live DSA key) for join to current Group C
2. $M_i \in C$ knows $ M_{n+1} $ or gets to know him. If he thinks this member is trustful he either
   - sends an approval to C~~. (Multiple approvals could be required)~~ ,
   - or if the number of required approvals is reached, announce the current tree to M_{n+1}.
3. Every member
   - updates his binary key tree by adding new member node and new intermediate node.

- removes all keys and bkeys from the leaf node related to the sponsor to the root node.

4. ~~The sponsor M_s~~ M_{n+1} additionally
   - generates new share and computes all [key, bkey] pairs on the key-path.
   - broadcasts updated tree including only bkeys.
5. Every member verifies the plausibility of the received tree and computes group key using the updated tree.

Now M_{n+1} knows who is aware of this key. To verify those he has to contact everybody, because pseudo members could be possible.

D.2.2. Exchanging Data

This section outlines the method used to protect data being exchanged between all members. As above, all exponentiations are done modulo a particular 1536-bit prime, and g is a generator of that group, as indicated in the detailed description below.

Suppose M_i has a message to send to all group members (C):

- M_i:
  1. Uses the key tree to generate a shared secret (s) for C, and generates the sending AES key (ek) and the sending MAC key (mk), as detailed below.
  2. Picks a value of the counter, ctr, so that the tuple (s, ctr) is never the same for more than one data message.
  3. Picks last received or referenced message id as history.
  4. Generates signature (sig) of history + msg with short-lived DSA key.
  5. Computes T = (sender id, receiver id, ctr, AES-CTR$_{ek,ctr}$(history + msg + sig))
  6. Sends T, MAC_{mk}(T)
- Every $M_m \in C$:
  1. Uses key tree and receiver id to generate a shared secret.
  2. Uses sender id to compute receiving AES key (ek) and receiving MAC key (mk), as detailed below.
  3. Uses mk to verify MAC_{mk}(T)
  4. Uses ek and ctr to decrypt AES-CTR$_{ek,ctr}$(history + msg + sig)
  5. Verifies signature with known DSA key
  6. Review history

D.2.3. Leaving

After a member (M_d) is gone, the following steps have to be processed:

- Every member:
  1. updates key tree by removing the leaving member node and relevant parent node.
  2. removes all keys and blinded keys from the leaf node related to the sponsor to the root node.
- Sponsor M_s additionally:
  1. generates new share and computes all [key, bkey] pairs on the key-path.
  2. broadcasts updated tree including only bkeys
- Every member computes the group key using the updated key tree.

D.3. Details of the protocol

D.3.1. Unencoded messages

This section describes the messages in the TBG-OTR protocol that are not base-64 encoded binary.

D.3.1.1. TBG-OTR Query Messages

If a user (A) wants to start a TBG-OTR session he sends a message starting with ?TBG-OTR? optionally followed by an explanation. Two cases are possible:

1. There is already an active TBG-OTR session in the broadcast channel. ~~The last active member~~ One or more members of that group should send a TBG-OTR error messages, followed by a request message from user A to join that group.
2. There is no active TBG-OTR session in the broadcast channel. All present users who want to join should send a request message and user A acts as first sponsor for the first received request message. All other messages are handled as described below.

D.3.1.2. TBG-OTR Error Messages

Any message containing the string "?TBG-OTR Error:", optional followed by the group fingerprint, plus the character ":" is an TBG-OTR Error Message. The following part of the message should contain human-readable details of the error.

D.3.2. Encoded messages

This section describes the byte-level format of the base-64 encoded binary TBG-OTR messages. The binary form of each of the messages is described below. To transmit one of these messages, construct the ASCII string consisting of the nine bytes "?TBG-OTR:", followed by the base-64 encoding of the binary form of the message, followed by the byte ".".

For the Diffie-Hellman group computations, the group is the one defined in RFC 3526 with 1536-bit modulus (hex, big-endian):

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

and a generator (g) of 2. Note that this means that whenever you see a Diffie-Hellman exponentiation in this document, it always means that the exponentiation is done modulo the above 1536-bit number.

D.3.2.1. Data types

Bytes (BYTE)
: 1 byte unsigned value

Shorts (SHORT)
: 2 byte unsigned value, big-endian

Ints (INT)
: 4 byte unsigned value, big-endian

Multi-precision integers (MPI)
: 4 byte unsigned len, big-endian
: len byte unsigned value, big-endian
: (MPIs must use the minimum-length encoding; i.e. no leading 0x00 bytes. This is important when calculating public key fingerprints.)

Opaque variable-length data (DATA)
: 4 byte unsigned len, big-endian
: len byte data

Initial CTR-mode counter value (CTR)
: 8 bytes data

Message Authentication Code (MAC)
: 20 bytes MAC data

D.3.2.1.1. Public keys and signatures

TBG-OTR users have long-lived public DSA keys that they use for authentication (but not encryption). And short-lived public DSA keys for message authentication. Both should be never be linked and the short-lived key should only be verified over a deniable channel (e.g. OTR session). The current version of the TBG-OTR protocol only supports DSA public keys, but there is a key type marker for future extensibility.

OTR public DSA key (PUBKEY)

Pubkey consist of the type and the key parameters Pubkey type (SHORT)

- DSA public keys have type 0x0000

DSA public key parameters

- p (MPI)
- q (MPI)
- g (MPI)
- y (MPI)

TBG-OTR public DSA keys are used to generate signatures; different types of keys produce signatures in different formats. The format for a signature made by a DSA public key is as follows:

DSA signature (SIG)

(len is the length of the DSA public parameter q, which in current implementations must be 20 bytes, or 160 bits)

len byte unsigned r, big-endian len byte unsigned s, big-endian

D.3.2.1.2. Fingerprints

TBG-OTR public keys (DSA and DH) have fingerprints, which are hex strings that serve as identifiers for the public key. The fingerprint is calculated by taking the ~~SHA-1~~ first 20 bytes of the SHA-256 hash of the byte-level representation of the public key.

Public key fingerprint (FP)

First 20 bytes ~~SHA1~~ of SHA-256 data

D.3.2.1.3. Type-Length-Value record

Each TLV record is of the form:

Type (SHORT)

The type of this record. Records with unrecognized types should be ignored.

Length (SHORT)

The length of the following field

Value (len BYTEs)
(where len is the value of the Length field)

Any pertinent data for the record type.
Some TLV examples:
`\x00\x01\x00\x00` A TLV of type 1 (`\x00\x01`), containing no data (length = `\x00\x00`)
`\x00\x00\x00\x05\x68\x65\x6c\x6c\x6f` A TLV of type 0 (`\x00\x00`) with length 5 (`\x00\x05`), containing the value "hello" (`\x68\x65\x6c\x6c\x6f`).

D.3.2.2. Request Message

This is the first message of the AKE. Alice sends it to the group to request acceptance.

D.3.2.2.1. Format

Protocol version (SHORT)
The version number of this protocol is 0x0001.
Message type (BYTE)
The request message has type 0x01.
Sender id (FP)
The long-lived DSA fingerprint of the person requesting acceptance.
Receiver id (FP)
The public DH fingerprint of the group key.
Blinded key (DATA)
The public DH key of the person requesting acceptance.
Plaintext message (DATA)
An optional plaintext message which is displayed to all members.
Signature (SIG)
The DSA signature, using the long-lived DSA key, of everything from the protocol version to the plaintext message.
Signature (SIG)
The DSA signature, using the long-lived DSA key, of the blinded key. This signature is transfered along with the updated key tree.

D.3.2.3. Tree Announcement Message

If the number of approvals is reached minus one, the last member M_i which approves the new member broadcasts this announcement instead of his approval, and consist of:

- Initial configuration (DATA) of the group. If no configuration is set (0x00 00 00 00) the default values apply. Currently it is not allowed to change this value after

group initialisation. And if it does, it should trigger a protest. The currently defined TLV (type-length-value) options are:

  - Type 0: Number of required confirmations (Default=1). Needs to be always 0 or greater. If this number is equal or greater than the current number of members, a confirmation from all members is required.
  - ~~Type 1: Number of seconds after a sponsor should be considered as offline (Default=8~~

- All BK signatures (DATA). Order all long-lived DSA leaf signatures (SIG) from left to right followed by there corresponding key id (FB).

D.3.2.3.1. Format

Protocol version (SHORT)
: The version number of this protocol is 0x0001.

Message type (BYTE)
: The tree announcement message has type 0x02.

Sender id (FP)
: The public DH fingerprint of M_i.

Receiver id (FP)
: The public DH fingerprint of the new member.

Group id (FP)
: The public DH key fingerprint of the corresponding tree.

Top half of counter init (CTR)
: This should monotonically increase (as a big-endian value) for each message sent with the same (sender id, recipient id) pair, and must not be all 0x00.

Serialized tree with BK (DATA)
: Serialize the current key tree in preorder with every BK (DATA) followed by either 0x01 (BYTE) for leafes or 0x00 for empty nodes.

Encrypted message (DATA)
: Using the corresponding two-party key (sender/receiver), perform AES128 counter-mode encryption of the message.

Signature (SIG)
: The DSA signature, using the short-lived DSA key, of everything from the protocol version to the encrypted message.

2P Authenticator (MAC)
: The SHA1-HMAC, using the two-party (sender/receiver) MAC key, of everything from the protocol version to the encrypted message.

Authenticator (MAC)
: The SHA1-HMAC, using the corresponding group MAC key, of everything from the protocol version to the encrypted message.

D.3.2.4. Tree Update Message

The sponsor sends this message after a group modification (join, leave~~, timeout~~).

D.3.2.4.1. Format

Protocol version (SHORT)
: The version number of this protocol is 0x0001.

Message type (BYTE)
: The tree message has type 0x03.

Sender id (FP)
: The public DH key fingerprint of the sponsor.

Receiver id (FP)
: The public DH fingerprint of the new derived group key.

Previous receiver id (FP)
: The public DH fingerprint of the group key which triggered this tree update.

Serialized tree with BK (DATA)
: Serialize the new key tree in preorder with every BK (DATA) followed by either 0x01 (BYTE) for leafes or 0x00 for empty nodes.

~~**Encrypted BK signatures (DATA)**~~
: ~~Order all~~

Configuration authenticator (MAC)
: The SHA1-HMAC, using the new MAC key, of the initial configuration.

BK signature authenticator (MAC)
: The SHA1-HMAC, using the new MAC key, of all serialized long-lived DSA leaf signatures (SIG) from left to right followed by there corresponding key id ~~, encrypt it using AES128-CTR and encode it as DATA field. The initial counter is a 16-byte value whose first 8 bytes are the above "top half of counter init" value, and whose last 8 bytes are all 0x00~~(FP).

Signature (SIG)
: The DSA signature, using the short-lived DSA key, of everything from the protocol version to the ~~encrypted signatures~~BK signature authenticator.

Authenticator (MAC)
: The SHA1-HMAC, using the new MAC key, of everything from the protocol version to the encrypted message.

D.3.2.5. Data Message

This message is used to transmit a private message to the correspondent, and consists of:

- MAC of the referenced or last received message (MAC).

- Plaintext message consists of a human-readable message (encoded in UTF-8, optionally with HTML markup) (DATA).
- Zero or more TLV (type/length/value) records (with no padding between them).
- DSA signature (SIG) with short-lived key of the first 3 items.

The currently defined TLV records for data messages are:

Type 0 Padding
: The value may be an arbitrary amount of data, which should be ignored. This type can be used to disguise the length of the plaintext message.

Type 1 Disconnected
: If the user requests to close the private connection, you may send a message (possibly with empty human-readable part) containing a record with this TLV type just before you discard the session keys, and transition to MSGSTATE_PLAINTEXT (see below). If you receive a TLV record of this type, you should inform the user that this member has closed his end of the private connection.

Type 2 Confirmation
: This type represents the confirmation of a join request and starts the tree update if the number of confirmations is greater or equal to the configured value.

~~**Type 3 Public DSA key**~~

Type 3 Public short-lived DSA key
: If a user sends a data message the first time after a new user joined, he has to send his public short-lived DSA key (PUBKEY).

~~**Type 4 protest**~~

Type 4 Public long-lived DSA key
: A freshly joined user has to send his public long-lived DSA key immediately after the join completed (PUBKEY).

Type 5 Protest
: If a fraud attempt is detected, every client should protest. If a protest message is received and believable, the affected message should be discard and possible key updates revoked. The value of a TLV type 4 consists of a reason (SHORT) followed by the affected message id (MAC). Reasons for a protest are:

  - 0x00 Other
  - 0x01 Tree manipulated
  - 0x02 Configuration changed unauthorized
  - 0x03 Member uses multiple short-lived DSA keys

Type 6 GPV request message
: The value represents an initiating message of the group parameter verification process, described below.

Type 7 GPV response message
: The value represents the final message in the group parameter verification process, described below.

Type 8 GPV abort message
: If one party encounters an error in the protocol, he has to send an group parameter verification abort message.

D.3.2.5.1. Format

Protocol version (SHORT)
: The version number of this protocol is 0x0001.

Message type (BYTE)
: The data message has type ~~0x03~~0x04.

Sender id (FP)
: The public DH ~~key~~ fingerprint of the ~~sponsor~~sender.

Receiver id (FP)
: The public DH fingerprint of the group key or a member.

Top half of counter init (CTR)
: This should monotonically increase (as a big-endian value) for each message sent with the same (sender id, recipient id) pair, and must not be all 0x00.

Encrypted message (DATA)
: Using the corresponding ~~group~~ receiver key (receiver id), perform AES128 counter-mode encryption of the message.

Authenticator (MAC)
: The SHA1-HMAC, using the ~~group~~ receiver MAC key, of everything from the protocol version to the encrypted message.

D.3.3. Group parameter verification (GPV)

To ensure that the group parameters (participants, tree) are correct, a member can verify those with as many other members (pairwise) as he likes, if his message state machine has MSGSTATE_ENCRYPTED set. All GPV messages must contain a Type 3 and 4 TLV.

D.3.3.1. GPV request message

With the tree verification request (Type 6 TLV) a member can start verification of his view about the group, it consist of:

- The public DH fingerprint of the group key (FP).
- The SHA1-HMAC (MAC), using the group MAC key, of :
- The previous group fingerprint (FP).

- All serialized long-lived DSA leaf signatures (SIG) from left to right followed by there corresponding key id (FP).
- The public short-lived DSA fingerprint (FP) of the receiver.
- The public long-lived DSA fingerprint (FP) of the receiver.

D.3.3.2. GPV response message

With the tree verification response (Type 7 TLV) a member can (and has to, if the MAC was correct) confirm that the previous received authenticator matches his view of the group, it consist of:

- The public DH fingerprint of the group key (FP).
- The SHA1-HMAC, using the group MAC key, of :
- The previous group fingerprint (FP).
- All serialized long-lived DSA leaf signatures (SIG) from left to right followed by there corresponding key id (FP).
- The public short-lived DSA fingerprint (FP) of the receiver (requestor).
- The public long-lived DSA fingerprint (FP) of the receiver (requestor).

D.3.3.3. GPV abort message

If one party encounters an error in the protocol, it should send an TLV type 8, possibly with empty human-readable part.

D.3.4. Recommendations for action

This section contains recommendations for action to handle special cases and improve security where needed.

D.3.4.1. Joining

- If a user sends a query message and multiple different groups exist, every group should send one error message.
- To reduce the number of error messages, every member should delay the sending of this message by a random time between 0 and 3 seconds. If they receive another error message in this time, they should abort the transfer.
- Members should wait a reasonable time until an ongoing join event finished with a tree update message, before the next tree announcement message is send.

D.3.4.2. Leaving

- If the sponsor, who is required to finish a leave request, is not responding, every remaining member should mark the leaving member and wait until the sponsor responses again.

- If a sponsor left a group, before he updated the tree, he should be marked and all members should check if they are the sponsor for both gone members.
- During an ongoing join process, nobody should leave a group.

D.3.4.3. Miscellaneous

- The number of pairwise group parameter verifications, depends on the level of confidentiality.
- Group parameter verification can be done automated in the background.
- No established member should initiate a group parameter verification with a new member.

D.3.5. Key Management

For each group a client should keep track of:

- At least 2 recent BK-Trees, until every member sends at least one message with the new key.
- Signatures of all public DH leaf keys.
- A mapping between each members used DH (leaf) and short-lived DSA key

The mapping between the members used DH key and their long-lived DSA key is done with the help of the list of all leaf signatures.

When starting a private conversation with a group, generate one DH key pair ~~for yourself~~and one short-lived DSA key for yourself, which does not change in all derived groups. Note that all DH keys should have a private part that is at least 320 bits long.

When you send a data message
: Use the root DH key of the desired group or member to calculate session keys, as outlined below. Use the "sending AES key" to encrypt message and the "sending MAC key" to calculate its authenticator.

When you receive a data message
: Use the ids in the message to calculate session keys, as outlined below. Use the "receiving MAC key" to verify the authenticator. If sender id is no leaf in the tree, the receiver is ~~is~~ no node or the MAC does not match, reject the message.

D.3.5.1. The blinded key tree

TBG-OTR uses Tree-based Group Key Agreement to calculate shared secrets:

- All members are leafes in a binary tree.
- Every node contains a blinded key (g^{s_i}) of a secret (s_i).

- All members need to share the same tree in order to calculate the same secret.
- To calculate the root secret, at least one private key is needed. At the beginning this is the private part of a leaf.
- With the known secret nearest to the root ($s_{i,0}$) and the public key of the neighbor node ($g^{s_{i,1}}$) the private key $s_{i-1,0}$ of the parent node is calculated with $g^{s_{i,0}s_{i,1}}$.
- Repeat the last step until you get the root secret (s).

D.3.5.1.1. Adding a member

To keep the tree compact, a new node (and therefore member; s_{n+1}) should be added to the rightmost shallowest leaf node (s_i). The member which owns the private part of node s_i or the new member s_{n+1} is called sponsor and has to perform the following actions:

1. Delete all blinded keys on the way up to the root ($s_{i..0}$).
2. Add the blinded key of his own node and from the new member as children to s_i.
3. Calculate all blinded keys up to the root, as described in the previous section.
4. Broadcast the updated tree with all blinded keys.

D.3.5.1.2. Removing a member

If a member (m) has to be removed, the neighbor of him is called the sponsor (s_i) and has to perform the following actions:

1. Delete all blinded keys on the way up to the root ($s_{i..0}$).
2. Replace the parent node with his node and remove the gone member m.
3. Calculate all blinded keys up to the root, as described in the section above.
4. Broadcast the updated tree with all blinded keys.

D.3.5.2. Computing AES and MAC keys

Based on the root secret s each member computes 2 values:

- One 128-bit AES encryption key code
- One 160-bit SHA1-HMAC key m

These keys are calculated as follows:

- Write the value of s as a minimum-length MPI, as specified above. Let this (4 + len)-byte value be "secbytes".
- For a given salt, define h1(salt) to be the 160-bit output of the ~~SHA-1~~ SHA-256 hash of the salt followed by secbytes.
- The "sending AES key" is the first 16 bytes of h1(own public DH key id).

- The "sending MAC key" is the 20-byte ~~SHA-1~~ SHA-256 hash of the 16-byte sending AES key.
- For every other member the "receiving AES key" is the first 16 bytes of h1(their public DH key id).
- For every other member the "receiving MAC key" is the 20-byte ~~SHA-1~~ SHA-256 hash of the 16 byte receiving AES key.

D.3.6. Fragmentation

Some networks may have a maximum message size that is too small to contain an encoded TBG-OTR message. In that event, the sender may choose to split the message into a number of fragments. This section describes the format of the fragments. All TBG-OTR clients must be able to assemble received fragments, but performing fragmentation on outgoing messages is optional.

D.3.6.1. Transmitting Fragments

If you have information about the maximum size of message you are able to send (the different IM networks have different limits), you can fragment an encoded TBG-OTR message as follows:

- Start with the TBG-OTR message as you would normally transmit it. For example, a Data Message would start with "?TBG-OTR:AAED" and end with ".".
- Break it up into sufficiently small pieces. Let the number of pieces be (n), and the pieces be piece[1],piece[2],. . . ,piece[n].
- Transmit (n) fragmented messages with the following (printf-like) structure (as k runs from 1 to n inclusive): ?TBG-OTR|%x|%x,%hu,%hu,%s," , sender id, receiver id, k , n , piece[k]
- Note that k and n are SHORT, and each has a maximum value of 65535. Also, each piece[k] must be non-empty.

D.3.6.2. Receiving Fragments:

If you receive a message containing "?TBG-OTR|" (note that you'll need to check for this *before* checking for any of the other "?TBG-OTR:" markers):

- Parse it as the printf statement above into k, n, and piece.
- If the recipient's id does not match any node, and the listed receiver id is not zero, the recipient should discard the message and optionally pass along a warning for the user.
- Let (K,N) be your currently stored fragment number, and F be your currently stored fragment. (If you have no currently stored fragment, then $K = N = 0$ and $F =$ "".)

- If $k == 0$ or $n == 0$ or $k > n$, discard this (illegal) fragment.
- If $k == 1$:
  - Forget any stored fragment you may have
  - Store (piece) as F.
  - Store (k,n) as (K,N).
- If $n == N$ and $k == K + 1$:
  - Append (piece) to F.
  - Store (k,n) as (K,N).
- Otherwise:
  - Forget any stored fragment you may have
  - Store "" as F.
  - Store (0,0) as (K,N).

After this, if $N > 0$ and $K == N$, treat F as the received message.

D.3.7. The protocol state machine

A TBG-OTR client maintains separate state for every group. This state consists of two main state variables (message and authentication) and some other information. Additionally a client has to maintain keys, verification state a.s.o for every group member.

D.3.7.1. Message state

The message state variable (msgstate) controls what happens to outgoing messages typed by the user.

It can take the following values:

MSGSTATE_PLAINTEXT
: This state is the initial state and the only way back to it is through a explicit user request (UI operation), to prevent revealing information by mistake.

MSGSTATE_ENCRYPTED
: This state indicates that all outgoing messages are encrypted and is reached after ~~successfull authentication.~~ successful authentication. Unencrypted incoming messages should be clearly separated from the encrypted messages or marked in distinct way.

D.3.7.2. Authentication state

The authentication state variable (authstate) indicates the process of the membership negotiation. It can take the following values:

AUTHSTATE_NONE
: This is the initial state and indicates that no authentication is in progress. ~~It is also the only state in which a new authentication progress is allowed to start. If multiple requests arrive they should be handled one after another.~~

AUTHSTATE_AWAITING_CONFIRMATION
: After a new member sends the join request all current members enter this state.

AUTHSTATE_AWAITING_TREE
: After the ~~required confirmations were~~ tree announcement message was send all current members enter this state. The new member enters this state right after he sends his request.

After the tree update was received:

- All members enter the state AUTHSTATE_NONE.
- The new member sets his msgstate variable to MSGSTATE_ENCRYPTED.
- The next join request is processed.

If there is a reasonable time between two transitions and there is another pending join request, it is allowed to start processing the next request which automatically terminates the existing request.