1. CONCEPTS FOR IMPLEMENTATION

In chapters above we discussed some possible security concepts and increments for the service location protocol. In this section we will introduce our practical solutions to fulfill the requested requirements.
First of all we need to establish security groups, where users can communicate using encryption. To create those groups we combined the already implemented SLP scopes with a group key agreement protocol. For this purpose we took the Tree-based Group Diffie-Hellman protocol (TGDH). TGDH is a group key agreement protocol which we use to create and share security group keys (details about TGDH will be introduced in section 1.2). We will describe why we picked this protocol and how we integrated it into the SLP.
Further we will discuss possible attacks on SLP which uses TGDH and the solutions we made to prevent them. (nachher etwas ausführlicher beschreiben)

1.1 Security Groups

Like discussed above we propose to create security groups in an open network to establish security between some users and still offer all SLP functions there. To create a secured group within an open network we worked out a concept and a new workflow for our SecuredSLP. First of all a security group in SecuredSLP is a SLP scope in which the group members use encryption to communicate with each other, which means the group members all share a secret. To create and share a secret we used the TGDH protocol. There are some requirements to a security group which we should look at:

- To join a secured group a user needs to know that there is such a group and this information should be announced as a plain text message, otherwise it isn't possible to get knowlege about such a group. The message should contain at least the information about the group name and where to get access to this group.
- After a user is allowed to join a group, he requires the shared secret of this group, the group key (see also sections 1.2 and 3.2).
- With a valid group key the user can decrypt all announced service information or offer services itself.
- If a user leaves a secured group, reasons could be for example an expired session key, the secured group key should be refreshed automatically.

We choose TGDH because it fulfill all the requirements above and we already found an implementation in java for it. But it is not necessary to use TGDH, so any other group key agreement protocol would work as well.
With TGDH each user within a secured group has several cryptographic keys.
überarbeiten!!!
Group Shared Key (GSK): This key is same for all group members and allows to access the secured group.
Private-/Public-Key: This is a key-pair that every user has and with those the users encrypt or decrypt their messages.

1.2 Tree-based Group Diffie-Hellman (TGDH)

Tree-based Group Diffie-Hellman is a protocol-suite for group key management. It handles the key distribution between all network members. TGDH is based on a binary tree structure. But the tree structure is just logical and isn't connected with the real position of the network members. This protocol-suite handles several dynamic group events like a new member joins or leaves the group and two networks merge together or one network partitions in several networks. So TGDH implements four protocols: *join*, *leave*, *merge* and *partition*[1]. But all of these protocols have some common structures with following features:

- Each network member computes the *group session key* out of the *group key* with a hash-function (the hash-function is the same for all members).
- Each *member session key* is just known to the member himself and shouldn't be published.
- In case a group gets larger, the *session keys* of new members will be included and some of old members have to refresh their *session key*.
- In case a group gets smaller, the *session keys* of the members, who left the group, will be deleted and at least one member of the group have to refresh his *session key*.
- All protocol-messages are signed by the sender. For the digital signature TGDH uses RSA or DSA with SHA-1 hash-function.

After any changes each member refreshes his *key-tree* independently from each other.

1.2.1 Keys in TGDH

There are several cryptographic keys which are used in TGDH.

Group key $K_{<0,0>}$ which is represented as the root of the TGDH tree (compare with figure 1.1).

Group session key K_{group} which is derived from the group key. $K_{group} = h(K_{<0,0>})$, where $h()$ is a cryptographic hash-function.

Member session key K_i is a session key for the member M_i.

Blinded key BK_i is a key which can be calculated from the member session key K_i with the function $BK_i = f(K_i)$, where $f() = g^k \ mod \ p$ with g as generator and p as a prime.

Furthermore there are also key sets a member has knowledge about. Each member knows all keys and their corresponding blinded keys in his path from the leaf to the root in the tree. For example member M_2 in the figure 1.1 knows the set of

[1]To secure SLP we only use the join and leave protocols. For that reason the other protocols won't be discussed in detail.

keys $\{K_{<2,1>}, K_{<1,0>}, K_{<0,0>}\}$ and the set of blinded keys $\{BK_{<2,1>}, BK_{<1,0>}, BK_{<0,0>}\}$. With that information it is possible to compute any other key:

$$\begin{aligned} K_{<l,v>} &= (BK_{<l+1,2v+1>})^{K_{<l+1,2v>}} \ mod\ p \\ &= (BK_{<l+1,2v>})^{K_{<l+1,2v+1>}} \ mod\ p \\ &= g^{K_{l+1,2v}K_{l+1,2v+1}} \ mod\ p \\ &= f(K_{l+1,2v}K_{l+1,2v+1}) \end{aligned}$$

It is also possible to compute the group key out of that information. For member M_2 the calculation would be:

$$\begin{aligned} K_{<0,0>} &= (BK_{<1,1>})^{K_{<1,0>}} \ mod\ p \\ &= (BK_{1,1})^{(BK_{<2,1>})^{K_{<2,0>}}} \ mod\ p \end{aligned}$$

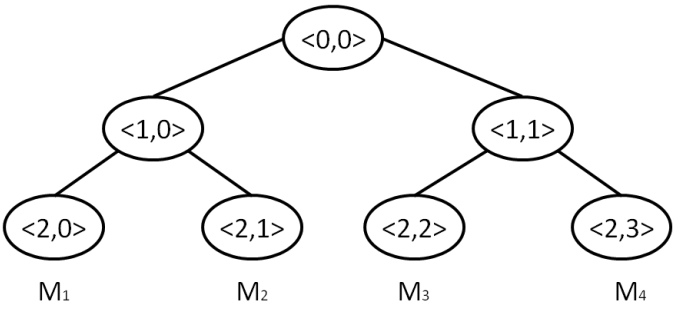

Figure 1.1: Example of a tree structure in TGDH

1.2.2 Join protocol

Assume a group with three members $\{M_1, M_2, M_3\}$ and a new member M_4 who wants to join this group. M_4 initialize the join protocol by sending a *JoinMessage* to the group via multicast. The message contains the blinded key of M_4. First of all the join position for M_4 is calculated and a *sponsor* is chose. The sponsor is the member whose leaf is placed on the insert position of the new member. In this case all members insert a new joint and a new leaf in their tree and delete all blinded keys in the path of the sponsor. Additionally the sponsor generates the new group session key and all keys and blinded keys in his path. Finally the sponsor sends the new tree $\widehat{T}$ and a set of all blinded keys via a multicast message. Member M_1 and M_2 can just calculate the group key after they received the new tree $\widehat{T}$. The tree update is shown in figure 1.2.

1.2.3 Leave protocol

To leave the group a member sends a *LeaveMessage* via multicast to the group. Same as in join protocol, a sponsor is chose. All members remove the leaf and his father joint from the tree. Also they remove all keys and blinded keys in the corresponding path. Additionally the sponsor generates new session key and computes all keys and blinded keys in his path. In the end sponsor sends the new tree $\widehat{T}$ and a set of all blinded keys via broadcast to the group. After receiving the new tree, all other members are able to compute the new group key. The corresponding tree update is illustrated in figure 1.3.

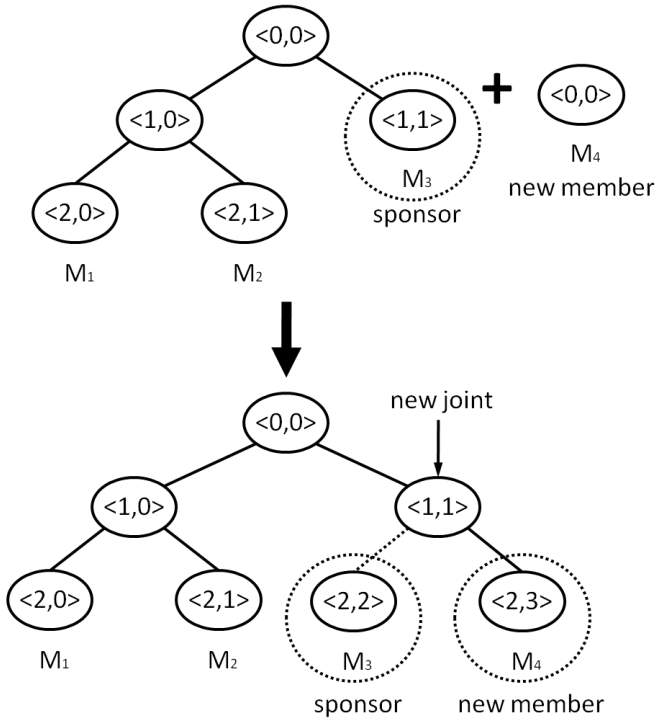

Figure 1.2: Tree update after a join of a new member

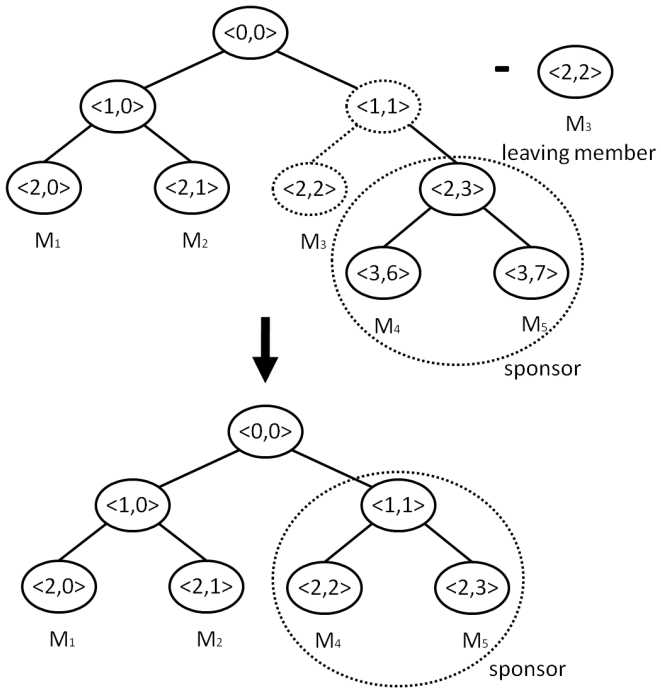

Figure 1.3: Tree update after a member left the group

2. INTEGRATION OF A GROUP KEY AGREEMENT PROTOCOL INTO SLP

There are two main approaches how to integrate a group key agreement protocol into the SLP. It is possible to integrate

the group key agreement hardcoded into SLP or as a module adjacent to SLP. Both got their benefits and disadvantages.

2.1 Hardcoded integration

With this solution we have to take the SLP source code apart and change many methods to combine a group key agreement protocol with SLP. That could be a neatly solution but would change the SLP too much and the implementation would be much complicated and expensive in time. The SLP probably wouldn't be compatible to the versions below and that isn't our goal. This approach would deliver a new protocol which probably would scale better as the modular integration solution. eventuell paar Einzelheiten ergänzen

2.2 Modular integration

With this solution we don't make many changes in the SLP source code. We use a group key agreement protocol as a module which we combine with SLP without fully integrates it into the SLP source code. In this way the module can easily be replaced anytime. So we are not bound to a special group agreement protocol. Also it is possible to use many several group agreement protocols at the same time if needed.
For this purpose we have minor changes in the source code. Like discussed above we took TGDH as our group agreement protocol. To make SLP work with TGDH we had to change the header of SLP messages and implement additional functions into SLP to communicate with our group agreement protocol. hier vielleicht mehr details

2.2.1 Workflow of SLP with a modular integration

hier noch ein Bild einfügen

3. IMPLEMENTATION

To combine SLP with our group key agreement protocol we changed following protocol properties:

- SLP message
-

3.1 SecuredSLP message format

The header of a SLP message was modified to distinguish SLP messages from SecuredSLP messages. We added the `S`, `Security Group Length` and `Security Group Name` flags into the header and changed the `Version` of SLP in the source code (compare figure 3.1 and figure 3.1).

Version: Is now set to 3.

S: This flag shows that the message body is encrypted.

Security Group Length: This flag specifies the length of the `Security Group Name` string.

Securety Group Name: This flag shows the name of the security group the message belongs to.

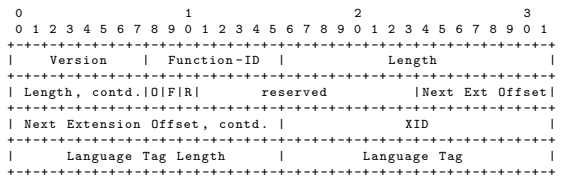

Figure 3.1: SLPv2 Header

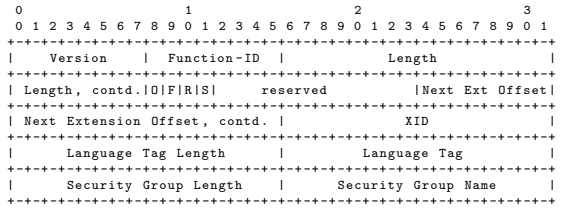

Figure 3.2: SecuredSLP Header

3.2 Key enchantment

To make a securety group work it is necessary to encrypt the messeges between secure scope members so nobody else can evedrop the network traffic. To encrypt messages every group member need to know a shared secret, a group session key. There are some solutions to generate and share a key in an open network like used in this work. We decided to use the Tree Group Diffie Hellman (TGDH) protocol. This decision was made by the following reasons:

- TGDH is a protocol designed for dynamic networks to enchante key between many group members
- TGDH can handle key enchantment between more then hundred members
- the key can be enchanted between the users over an unsecured channel
- and we found an implementation of TGDH in Java so we could use it without many troubles