INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Group Key Management with Strongswan

Wolfgang Engelbrecht

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Group Key Management with Strongswan

Wolfgang Engelbrecht

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Tobias Guggemos
Abgabetermin:	23. Januar 2018

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 23. Januar 2018

.....
(Unterschrift des Kandidaten)

Abstract

In recent years the Internet of Things (IoT) has shown a rapid growth which leads to many devices with limited resources being added to the internet. Many of those require communication on a group level to effectively manage their limited resources and provide maximum use of their presented information. The amount of keys that need to be managed by each device rises exponentially when using secured communication. A group key management in combination with a multicast architecture could solve this problem. The client need only to store the keys needed for the communication to the server and the groups they are subscribed to. The group key manager takes care of the authentication and authorization of the group members wanting to join and distributes the keys accordingly. It also manages the lifetime and policies that are in action in its managed groups. G-IKEv2 is a proposal for a standard to ensure such a thing based on the IPsec protocol IKEv2. This thesis implements the base of a group manager in the already established IPsec suite Strongswan.

Contents

1 Introduction

In today's society more and more devices are designed or upgraded with network capabilities. This is done to allow transfer of data collected by the devices. The collection of small networking devices which can communicate over the internet is commonly called the "Internet of Things" (IoT). The employed use cases range from sensors used for the monitoring of data concerning a single person to wide arrays of sensors used to analyze traffic or weather. But not only sensors are included in the IoT. Any device that can provide added functions by gaining network capabilities is a candidate for joining. This can be something simple like a refrigerator sending reports as to its contents. However the generated data has to be used and analyzed somewhere as many IoT devices were not designed for this. They simply share their data with a set of receivers who can use them. Vice versa they can also be reliant on the information passed from other devices. This leads to the logical structuring of devices depending on each other being sorted into groups. The communication within such a group can be very straining on the network being used due to a large amount of overhead involved in the transport of data. Classical infrastructure is based on host-to-host connections which is also called unicast. If this is to be used as a means of group communication each sender would have to establish separate connections for sending the same data. This also puts more load on any routing devices between sender and receiver that need to determine how to handle the network packets. This can however be solved by switching communication to a multicast architecture. Multicast allows the sending of a single message to a multicast-address and anyone subscribed to this address receives this message and can process it. It has some resemblance with a broadcast architecture, but only subscribed members usually process the received message, while the others discard it immediately. Furthermore if the network infrastructure supports this, the message can be directed towards the registered subscribers to the multicast. For this the routing hardware has to keep a list of affiliations of network members. Due to the decrease of messages needing to be sent the load on the network infrastructure decreases rapidly which solves the problem of clogging the network with multiples of the same message.

Another problem arising is the confidentiality of data. Some of the data that is being sent can be of a sensitive nature, like health data collected by wearable devices, and therefore should be transported securely. Most common needed features in this respect are determining the authenticity of sender and receiver and the encryption of the data so no one listening in on the communication can read the plain data. This leads to authentication and encryption data needing to be stored for each connection. This is no problem for devices with a higher amount of resources, but can pose a problem for IoT devices which are most of the time designed to be as lightweight as possible. And not only the storage of security related data is a problem in this respect also the time and power needed to perform calculation intensive encryption algorithms. Here again the multicast architecture provides a help since it can reduce the amount of security data stored to one set per group affiliation. This takes care of the limited memory of IoT devices. However the key needed for the secure group communication needs to be generated and distributed. This can be solved by including a

third party which manages the key generation and distribution. This third party has to fulfill several requirements. It has to manage policies, keys and certificates for all groups it manages. It also has to provide mechanisms to authenticate and authorize potential group members. To fulfill its role as group key manager it has to be trusted by all devices. For this one can employ certificates which would possibly be too big for IoT devices or other authentication mechanisms. After successful authentication a secured communication channel has to be established for the transmission of keys for the group communication. After authenticating a device, the key manager has to verify if the requesting party is authorized to join the requested group. There is the possibility inside a group to distinguish between senders and receivers which also has to be assigned correctly. For this authorization the key manager has to have a database correlating the provided credentials with capabilities. Upon successful authorization the group key manager informs the group member of the policies and algorithms used in the group and provides the keys needed.

As a solution for this there is a proposal for a Group Internet Key Exchange (G-IKEv2) protocol which addresses these points. It proposes a way to quickly establish a secure connection and manage group access including the distribution of needed keys. This protocol is based on the established Internet Key Exchange (IKEv2) protocol from the IPsec suite. This has so far only been sparsely implemented as a test in a router software by CISCO. This however is not feasible for the internet of things as the resource requirements are out of scale with a minimum requirement of 4GB of memory needed to start up correctly. For this thesis Strongswan was chosen as it already implements the IKEv2 protocol but misses any group key management functionality. Strongswan is open source and has no plans to include group communication from their side. Due to these facts it presented a good opportunity to introduce key management into this framework.

Structure of this thesis

Chapter 2 gives an overview of possibilities for securing network traffic and the basics of group communication and management.

Chapter 3 analyzes the requirements for group key management in the context of IoT.

Chapter 4 showcases example protocols which are designed to manage group key generation and distribution.

In chapter 5 the concept of the group key management system is shown in detail and the design choices which were needed for the successful integration of this protocol into Strongswan are displayed.

Chapter 6 then shows the implementation of the structures and the adaptations which needed to be made in the code of Strongswan.

In the following chapter 7 the implemented group key management system was tested to determine if the protocol was correctly implemented.

Chapter 8 takes another look at the accomplishments of this thesis and provides a list of tasks which should or could be addressed in the future.

2 Background

This chapter aims to give an overview of the basics concerning the security of connections, the communication within groups and concerning the distribution of group keys.

2.1 Security

The term security in the case of network traffic encompasses multiple functions.

- Authentication
- Integrity
- Confidentiality

Authentication is important as it verifies the identity of the communication partner. To perform this step there are different mechanisms. A pre-shared secret can be used in combination with a negotiated algorithm to provide proof of knowledge of this secret. The algorithm is needed as otherwise the secret would be transmitted openly allowing spying third parties to intercept and misuse it. Another possibility is the usage of asynchronous key pairs. This is commonly performed with the use of X.509 certificates [CSF+08]. The sender encrypts a message with its private key and the receiver can decrypt it with the use of the public key contained in the certificate. The receiver also has to ensure the validity of the certificate by querying the contained authority if the id matches. If needed the next authorities higher up in the hierarchy need to be queried until a trusted authority is reached.

In addition to the authenticity of the partner the integrity of the message is of importance. This can be verified with the use of hashing functions which if correctly chosen can be used to generate a short representation of the message. Upon receipt of a message it is entered into the previously negotiated hash function and the values compared to those included in the message.

If the data which is to be sent is of a sensitive nature the employment of encryption algorithms is of importance to stop unauthorized entities to listen in.

These three tasks can each be used separately but commonly are used in combination. They are part of nearly all security protocols. They can be implemented on different layers of the network stack which leads to several available protocols being available providing very similar functions. They do however differ in their requirements as operating on a lower layer might require more permissions on the system. For each of those functions different keys are used which need to be stored and associated with the correct communication partners. For this so called Security Associations (SA) are used. SAs commonly contain information concerning the used algorithms for encryption, authentication, integrity verification and the corresponding keys. In the case of some protocols the amount of keys to be stored differs depending on the methods used. In the case of the Internet Key Exchange (IKE) two keys are stored each for the encryption and authentication since one is used per direction of the

traffic. If the protocols require Security Parameter Indexes (SPI) they are also stored in the SA. Furthermore a SA contains information concerning the protocol which is to be used and the correct parameters for this. This can include the lifetimes of the keys or the way packets are to be built and handled. Each device has to handle several SAs with each one corresponding to a single connection. In the case of this thesis the concept of group security associations (GSA) is used which are also treated as single SAs but the contained information regarding the connection target are related to a group. Examples for security protocols are the security architecture for the internet protocol (IPsec) or the Transport Layer Security protocol (TLS).

2.2 IPsec

IPsec was created to provide security for IPv4 and IPv6 including access control, integrity checks, source authentication, encryption and replay detection [KS05]. It operates on the the same layer as the Internet Protocol (IP) and can therefore be used to provide the aforementioned protections to all protocols working over IP. It provides two transport protocols, the authentication header (AH) and the encapsulating protocol (ESP), which can be used alone or in combination. The latter option is seldom used since ESP fulfills most of the requirements on its own. When using ESP in transport mode the original IP payload is encrypted and the knowledge of the corresponding key is needed to allow access to the original payload. The only thing readable is the header to allow correct transmission of the packet. The ESP header itself is included in the encrypted payload. The transport mode is commonly used for host-to-host communication whereas the tunnel mode wraps the complete original IP packet with a new header and trailer and encrypts the original packet. Since the mechanism is similar to the workings of an IP-tunnel it is mostly used for gateway-to-gateway communication.

2.3 IKEv2

The internet key exchange protocol was devised to allow a secure key exchange between two communication partners. The process in version 2 [KHN+14] of the protocol is split into two message exchanges, each consisting of a request and a response message. The first exchange is the so called IKE-SA-INIT in which the initiator and the responder negotiate the algorithms to be used for the creation of a secure communication channel. The messages consist of a header, security association (SA) payload, key exchange payload (KE) and a random value (nonce). The header contains information needed for the correct interpretation of the message. This includes the version numbers of the protocol, the security parameter indexes (SPIs) and several flags. The SA payload contains the supported cryptographic suites for an IKE-SA. The KE payload contains the Diffie-Hellman value generated by the initiator. The responder then answers with a message of the same structure but containing different values. The nonce is a new random value, the SA payload contains a cryptographic suite which was chosen from the ones proposed by the initiator, whereas the KE payload contains the data needed by the initiator to finish. Upon completion of the Diffie-Hellman exchange both partners have a shared secret for later use [Res99]. After validation of the response the next exchange is initiated. The second exchange is the IKE-AUTH in which the two participants authenticate each other with the provided materials. This can be either

done by using pre-shared keys for encryption of the AUTH payload or with the employment of certificates which prove the initiators identity and the associated privat key was used to encrypt the AUTH payload, which can then be decrypted with the contained public key. The latter involves the presence of a validation structure for the certificates. After a successful authentication this exchange can also be used to create the first CHILD-SA. This CHILD-SA will from then on be used for the secure communication between the partners. This will be performed using keys derived from the secret created during the IKE-SA-INIT using the the negotiated mechanisms. Further CHILD-SAs can be created later on with the corresponding exchange. The CHILD-SA contains all keys needed for securing the following network traffic. As the keys are commonly distributed with a certain lifetime they have to be renewed periodically which is performed by using a rekey exchange.

2.4 Group communication

Group communication can be performed in different ways depending on the use case. In the case of IoT devices which to a big part run on batteries the energy consumption is an important factor. These devices also commonly communicate via wireless networks. Since the IEEE 802.11 standard which is mostly used for general computation devices requires a certain amount of energy this can lead to a higher maintenance effort for constrained devices. To prevent the unnecessary energy consumption a different protocol was proposed. The 802.15.4 standard [GCB03] proposes the integration of networks with a decreased polling rate in comparison to 802.11 [IEE16]. This decreases the available speed and range but also reduces the energy consumption which makes them favorable for the use with IoT devices. This standard takes care of the two lowest layers in the network stack. On the network layer it was suggested to use IPv6 since it provides a better scalability in comparison with IPv4. However the header size can also decrease the amount of data which can be sent, so it was suggested to compress the IPv6 header to allow increased payload size. The Transport Control Protocol (TCP) is seldom used as the missing handshake in the User Datagram Protocol (UDP) produces less overhead further decreasing energy consumption. UDP does not provide assured delivery in itself. If this is required it has to be managed by the application. On the application of the layer the Constrained Application Protocol (CoAP) has been defined [SHB14] and it has since then become widely used. Its structure is similar to that of the hypertext transfer protocol (HTTP) regarding the message types integrated (GET/POST), however the headers are kept minimal with a fixed size of 4 Byte. It provides the possibility to designate UDP messages as confirmable which then prompts resending until an acknowledgment by the recipient is received. If this is not the case as can be the case with sensors the data is sent without retransmission. This allows the senders to stream their data by multicast to save bandwidth and battery life, whereas important messages can be confirmed by the listener. If the communication should be secured the usage of DTLS has been proposed [RD14]. The system is designed to use a group security association for the multicast message and additional SAs for the responses from the listeners, as the protocol requires the responses to be secured. Even though it was designed for IoT devices this uses a lot of resources of senders and listeners. The process also needs many message exchanges since every unicast SA needs to perform a handshake exchange. To alleviate the need for performing a handshake for each unicast connection between sender and listener Axiom [TNR17] was designed. In Axiom the individual keys needed for securing

the unicast responses are derived from the common group key material removing the need for DTLS handshake exchanges for this task.

There has also been a proposal as to the use of CoAP to manage group communication [IHA+14]. In group CoAP the group members send the messages which concern the group to a central entity which analyzes the message and the proceeds to send unicast messages to each group member. This is meant to take the load off of the single devices. It also was proposed as usable without requiring an infrastructure capable of managing multicast messages. The central entity performs a service similar to the one which would be provided by a network routing entity handling a multicast message. Here also only one set of keys is needed for the communication to the group and one for the managing entity. The managing entity has to maintain a list of devices registered in the managed groups. This means that the central entity takes on the role of group manager. In such a centralized structure the group members need to register themselves at the group manager or the group manager has to implement a database containing information about possible group members. Otherwise a mechanism needs to be provided to detect possible group members on the basis of provided specifications. The group manager controls access to the group by identifying the members and then authenticating and authorizing them should confidentiality of the group communication be required. The option of a decentralized group management system is not applicable in the scope of IoT devices as it would require too much resources.

3 Group Key Management

This chapter aims to analyze the demands which have to be met when implementing a group key managment from the perspectives of security and the usage in the IoT environment.

3.1 Security

If security is required in the group the management of keys is needed. In the case of multicast capable groups the Multicast Security (MSEC) group key management architecture was proposed [BCDL05]. It provides requirements and the design which should serve as the basis for the creation of centralized group key management systems. Such a system should include a protocol for the registration and de-registration of group members. This system should be unicast based and create a secure communication channel between member and server and allow the transfer of needed keys. This should be done by authenticating each other and then confidentially supplying the data needed for the creation of SAs for group internal data transfer and the rekeying process. Also a rekeying protocol should be defined as keys may be changed upon group membership changes, changes in policies, key changes or simple key expiration. The validity of a rekeying message can be either provided by the key manager including a means of source authentication or by the use of a authentication on group level. The latter requires the trust that no group member apart from the key manager sends rekey messages. The systems structure is shown in figure 3.1.

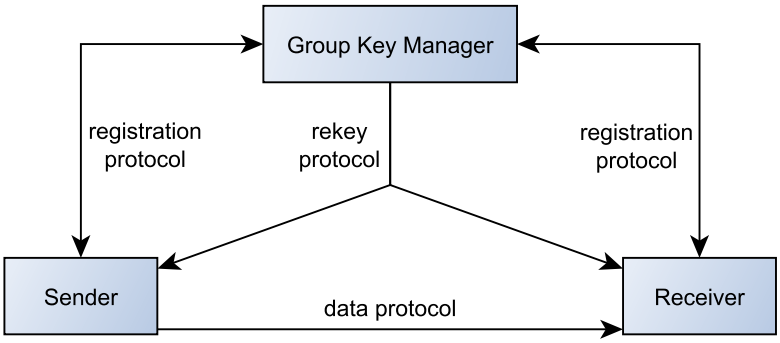

Figure 3.1: Key management system proposed by the MSEC draft. Figure adapted from [BCDL05]

3.2 Use Case

The planned integration into IoT networks requires the availability of clients capable of running on constrained devices without heavily impacting performance. A minimal client

implementing only the bare necessities for the realization of a protocol would fulfill this. This also favors a centralized group key management system as it allows the outsourcing of resource intensive tasks to a manager with more available resources. The necessary number of message exchanges for group registration and key distribution should also be kept to a minimum to reduce impact on the battery life of constrained IoT devices. This also includes the usage of a multicast solution to allow the distribution of new keys with a single message by the key manager.

3.3 Requirements summary

The following list aims to show the requirements on a group key management system based on the previous two sections in a concise matter.

R1 Data replay protection: The GCKS should be able to recognize the multiple receipt of the same message and act accordingly i.e. by discarding the additional messages. This stops a malevolent third party from being able to impersonate a different entity.

R2 Group level data confidentiality: The data sent within the group has to be kept secret and unreadable for members outside of the group. This might not be required if the data is not confidential but should be provided to allow a complete implementation of security.

R3 Group level authentication: The single group members need a way to determine if the received message originates from a valid group member which is important in dynamic groups.

R4 Data integrity: The validation of data integrity also has to be managed by the GCKS by providing the group members with the algorithms and keys to use for this task.

R5 Establishment of a GSA: The GCKS has to be able to establish a GSA which is fitting for the group. This includes selection of fitting algorithms for the encryption, authentication and data integrity verification.

R6 Distribution of a GSA: The GSA has to be securely distributed to all group members which is normally done by using a secure channel established prior to the registration process.

R7 Re-keying: Allow the distribution of new keys upon group events or the expiration of the keys lifetime.

R8 Backward and forward secrecy: In the case of dynamic groups the forward and backward secrecy should be able to be upheld. This requires rekeying upon the joining or leaving of a group member. This might involve the integration of more complex mechanisms for rekeying to allow for the system to remain scalable.

R9 Scalable group size: The protocol and the internal structure of the group manager should be able to scale with an increasing number of group members without suffering from severe performance losses.

R10 Low number of required messages to securely distribute group keys.

R11 Multicast communication: The group key manager should to be able to support the use of multicast based groups. This includes usage of multicast messages for the management of group functions such as the distribution of keys.

R12 Independent of layer: The key exchange should allow the distribution of keys and policies independent of the targeted layer of the following group communication.

In addition to these requirements the key manager also has to handle basic group manager functionality. It controls access to the group and manages members by changing their roles if applicable or revoking or granting member status.

4 Related Work

This chapter will provide a short overview of group key management designs and their basics.

4.1 Group DTLS

The probably most well known and used security mechanism for data transfer is the transport layer security (TLS) protocol described in [DR08]. It is used when accessing web addresses with a browser or when accessing email services. It provides end-to-end encryption of data on the session layer of the network stack thereby providing a way to exchange confidential data. TLS relies on the underlying connection to be managed with the transport control protocol (TCP) [Pos81]. This ensures the correct order of packets during the initial message exchanges. In these first messages the keys needed for the secure communication are negotiated. TCP is not feasible to use for group communication since the needed establishment of a connection as part of the initial TCP handshake is not capable of using multicast.

To allow the usage of a TLS based protocol the datagram transport layer security (DTLS) protocol was created [RM12]. It expands the TLS protocol with additional fields to correctly associate received packets. The combination of a sequence number, which determines packet order, and a so called epoch field, which identifies the exchange the packet belongs to, can be used as a unique identifier for packets. To account for packet loss DTLS also introduced a retransmission after a certain delay when no response is received. These changes were necessary to allow the usage of UDP as transport protocol. This enables the use of a multicast architecture. However to allow the protocol to be used for a group key exchange it had to be expanded one more time with a header preceding the standard DTLS header. This header is used to sort packets into a logical traffic system called channels. These channels include a client channel which is to be used for registration and deregistration purposes and other communication from the client to the group controller. An election channel can be used to elect a new group controller if this is permitted and the active controller fails. Key distribution and rekeying operations are handled on the control channel. The entity structure and the needed protocols are based on the ones proposed in [BCDL05] as registration and rekeying protocols are integrated and the general order of tasks is used. To join a group a potential group member has to first establish a channel to the group controller. The structure of the exchange is shown in 4.1. Here the stacked messages originating from the different protocols are made visible with the use of the boxes and the corresponding labels on the left which designate the origin of the exchange. This is done by performing a handshake comprised of six messages. This handshake is performed with the the client id in the additional headers being randomly generated prior to the first message. The client starts with a message indicating it wants to establish a connection. The controller then sends a response containing possible certificate, key exchange payload and certificate request. This is in turn answered by the client with its certificate, key exchange and a certificate verification payload. After this exchange has finished both parties generate a shared secret form the exchanged data. With this the client channel has been established. The client then requests

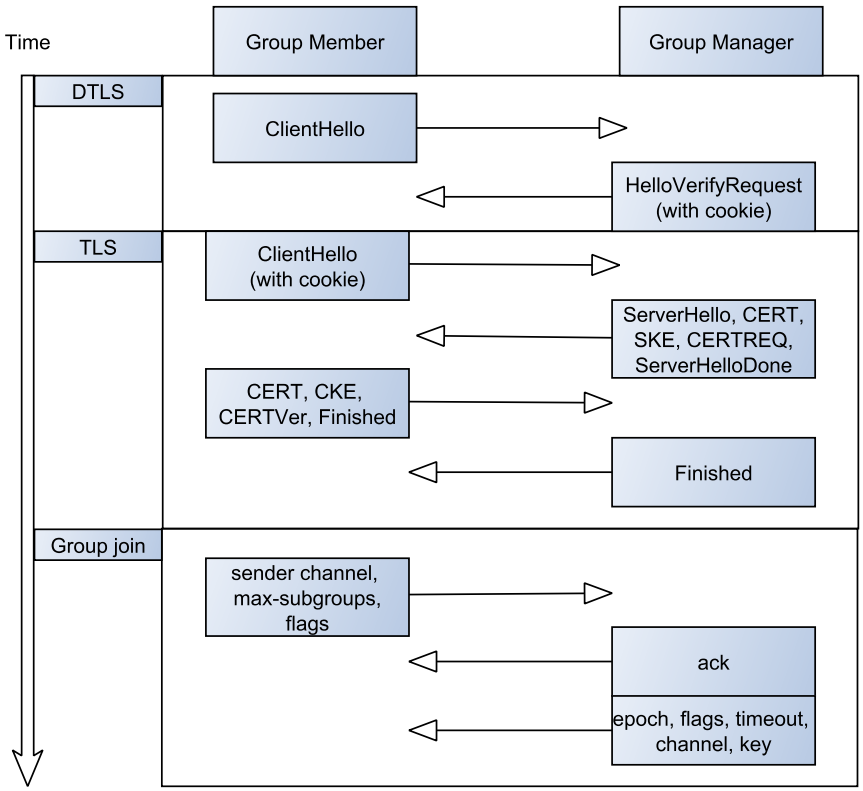

Figure 4.1: Messages needed for the secure key exchange using DTLS. The boxes on the left show the origin of the protocol parts

the keys to decypher group messages. If this is acknowledged by the controller the client is sent the key else the connection is closed and no key is sent. Should the connection established during this initial exchange be deleted or the client not answering on heartbeat messages the connection is closed by the controller and a key rotation for the other group members initiated. The amount of messages needed until the distribution of the group keys is higher than needed as each expansion of the protocol on its derivation from TLS added more. This is not in concordance with the requirement defined in R10.

4.2 Distributed Key Generation

Another possibility for the generation of group keys is the logical extension of the Diffie-Hellmann exchange to encompass more than two partners. To accomplish a distributed key management system each member has to keep an internal structure to represent the current state of the group. Each member contributes a part of the key. The group members are sorted into a binary tree structure to allow the association of the individual partial keys. This however requires the messages to be source authenticated. Upon joining the new member is entered into the appropriate place in the tree and announces its own partial key in a blinded fashion. Then the other partials keys are distributed in clusters to reduce overhead and a

new group key is generated by each member. This works since it has knowledge of its own partial key which is integrated into the combined key clusters. Upon leaving of a group member the previous neighbor node creates a new partial key and changes its position in the binary tree. Then the path to the tree root is reevaluated concerning the partial keys and the combined key sent to the rest of the group members. Then each group member creates the new group keys. A more detailed explanation of this mechanism can be found in [KPT00]. This mechanism needs a lot of managment by each group member as it needs to restructure its internal representation of the group upon each group change. This high impact on the resources of the devices makes it unfeasible for the use with constrained devices. It also is limited in its scalability due to the key calculation algorithms. The last two points are in conflict with R9.

4.3 GDOI

Another protocol which can be used for the distribution and management of group keys is the Group Domain of Interpretation (GDOI) [WRH11] which is also based on the MSEC structure. In contrast to the application layered protocol based on DTLS it was developed on the basis of the IKEv1 protocol [HC98] to allow secure key exchanges. The establishment of a secure connection between a group member and the group key manager is performed using a standard IKE setup. The structure of the message exchanges is shown on 4.2. The initial exchange is comprised of a exchange of SAs with the initiator proposing several policies and the responder answering with its choice. Then a Diffie-Hellman exchange is performed using the algorithm negotiated for the key derivation. After this the identifications are exchanged together with hashes used for authentication. This concludes the IKE exchange which now provides the secure channel needed. The group member now starts a pull request for the keys. Each message uses a different hash function including fields extracted from a previously received message. The first message initiates the request and provides the group identification to join and a nonce. The group manager answers with the SA containing the policies for the group and its own nonce. After another message by the group member which can contain a request for certain policies the group manager distributes the keys and optionally a sequence number payload indicating the last sequence number used in the group. Should new keys have to be distributed to the group this is done via multicast and encrypted using the KEK. However the high amount of message being sent contradicts the requirement set in R10.

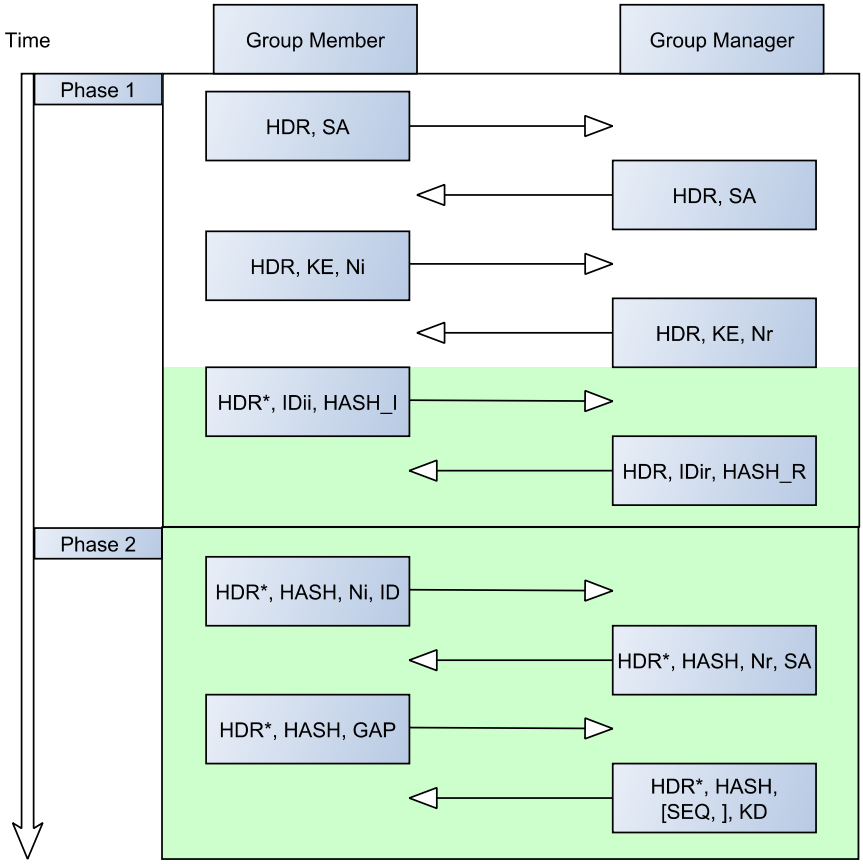

Figure 4.2: Messages needed for the secure key exchange using GDOI. Green background shows the encrypted messages.

5 Design

This chapter focuses on the design choices taken to allow the management of groups and the respective keys. The first section will showcase a group key management based on the proposal by [WNS17]. Then differing design choices will be highlighted which were taken in the scope of this thesis.

5.1 Architecture

The setup of a group with group based communication is comprised of several roles. The group controller/key server (GCKS) and group members (GM) which can be divided into senders and receivers if the protocol distinguishes between these two roles. The group members use the information provided by the GCKS for secure communication within the group.The GCKS has to ensure all devices with knowledge of the current keys are authorized. This allows group members to take the authenticity of received messages encrypted with the group keys as granted. In the proposed architecture sender authenticity is not inherently available during group communication due to the use of multicast. Since authentication and authorization are performed by the GCKS it can at least be determined that the message has been sent by an authorized member of the group. This is included in the G-IKEv2 protocol as the authentication and authorization of potential group members is a integral part of the two phase exchange upon registration. G-IKEv2 was proposed by [WNS17] and serves as a base for this thesis as it supports these requirements and uses fewer messages than other group key exchange protocols previously mentioned. Figure 5.1 shows a comparison to its predecessor GDOI. The lower number of exchanges originates from the combination of the

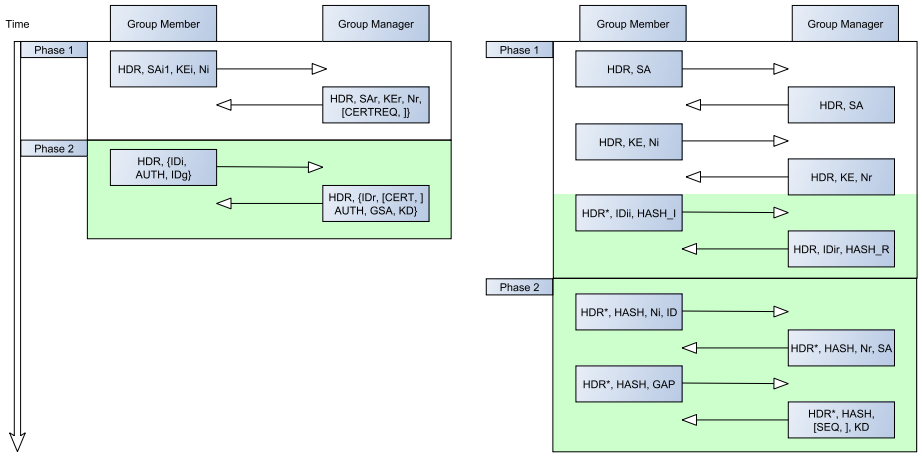

Figure 5.1: Comparison of exchanges needed by G-IKEv2 and its successor GDOI

authentication exchange and the registration exchange.

The process to join a group is initiated by establishing a secure communication between the prospective group member (GM) and the GCKS. The address of the GCKS has to be provided as it does not announce its provided service. The messages needed for this exchange are explained in further detail in 5.2. Due to the merging of authentication and registration the securing of the channel overlaps with the group join request. At this point the GCKS completes its authentication of the GM and retrieves the rights associated with the GM. It then responds with the appropriate information depending on the rights it found. This includes keys for encryption, authentication, integrity checking and the policies concerning the algorithms these keys are associated with. Further information provided is the lifetime of the keys and policies and information needed for the creation of a rekeying SA if the registration SA is to be destroyed after a successful distribution of group keys. Rekeying is needed to periodically refresh keys and to allow the integration of forward and backward secrecy. After the successful distribution of security information the GM now possesses the information to communicate within the group.

A standard rekeying process in G-IKEv2 distributes new keys (which can include keys for encryption, authentication and integrity verification) by sending a multicast message to the group with the use of the previously distributed KEK. Each time a rekeying is performed the message id used for the next rekeying is increased. Numbers lower than the one used at the last rekeying are to be ignored by the client. The system proposed by RFC4046 proposes a de-registration procedure initiated by group members. However G-IKEv2 does not include such a function. Here only the GCKS can remove members from the group. This can be done with the help of a message requesting the deletion of the GSA, which should be followed by the initiation of a rekeying of the remaining group members. This is required if forward secrecy is required to prevent the decoding of messages after leaving the group. This process is more resource intensive than a standard rekeying as the old KEK cannot be used.

The remaining members are to receive new KEKs before a rekeying using the new KEKs is initiated. This can be managed in different ways with differing resource intensity. The simplest case would be using unicast messages using the connection which was used for the initial key distribution or a rekeying SA which could be established during registration. However this is very resource intensive which makes it feasible only in groups with limited size and few changes in memberships. A more complex option would be the usage of a logical key hierarchy (LKH) for the key management which should be supported by the GCKS to allow a scalable group management. A LKH reduces the amount of key replacements needed in the case of leaving of joining members. A LKH views the members of a group as nodes in a tree structure with different KEKs per level and subbranch. Using this the amount of KEKs that need to be changed is drastically reduced. With the inclusion of a LKH the manageable group size drastically increases as rekeying to provide forward secrecy becomes less resource consuming.

The topic of backwards secrecy is easier to solve. In this case the rekeying is performed as soon as a new member joins. This also changes the registration process a bit in that it now does not include the TEK upon registration. Only the KEK is sent to the group member which then receives the TEK at the same time as the rest of the group via a rekeying message. The TEK provides data confidentiality on a group level as only members of the group know the keys to decrypt data payloads. In the case of G-IKEv2 the encryption keys are meant for symmetrical encryption and therefore allow no distinction between senders and receivers. The adaption to an asymmetric system however is easily included. To allow all these operations the GCKS has to keep an internal database containing information on the

groups it manages. The entries contain the roles each group supports, the keys, supported algorithms, policies and lists of possible members with their associated roles. This database has to be initialized before the first group members can join, preferably on startup. The database can either be kept consistently or regenerated from separate configuration files on each start. This member information in the group database is associated with the typical databases used for IPsec, which will be elaborated on in a later section, to decouple group specific information from general information needed for the initial exchanges.

5.2 Messages

This section describes structure of the message exchanges performed in the process of joining a group managed by a GCKS for G-IKEv2. The process is split into two phases of exchanges, each consisting of a request and a response. The first phase is performed the same way as the INIT phase of a standard IKEv2 initialization, therefore it will simply be called the IKE-INIT. This phase resembles the part of establishing a trusted connection between the prospective GM and the GCKS which is required to comply with the rules proposed by the MSEC. In this phase the GM send a request to the GCKS containing a list of supported cryptographic algorithms (SAi), the initiator's Diffie-Hellman value in the key exchange payload (KE) and a random number called a nonce. The header contains other data needed like the security parameter indexes (SPIs), protocol version numbers or exchange type. The GCKS chooses an algorithm from the proposed ones that matches its own criteria and informs the GM by including the chosen one in the SAr payload of the response. The response also contains the GCKS's Diffie-Hellman value in the KEr payload and its own nonce. Optionally a certificate can be requested from the GM for authentication. After this exchange is completed both sides can derive a shared secret from the Diffie-Hellman exchange which is used to derive the keys used for further communication. This is shown in 5.2. At this point the prerequisite of a secure connection is given but no authentication has taken place.

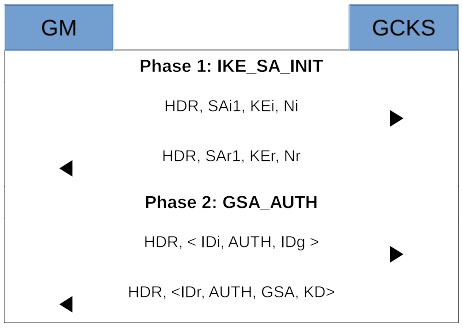

Figure 5.2: G-IKEv2 messages

In the next phase, called the GSA-AUTH, GM and GCKS authenticate each other and manage the registration to a group. The GM sends a request containing authentication data and the identification of the group it wants to join. The message consists of a unencrypted header and a payload encrypted with the algorithm and key negotiated in the initial exchange. This encrypted payload itself contains several payloads needed for the process. The GM provides an identification payload (IDi) and an authentication payload. If a certificate

was requested by the GCKS it also has to be included in this step. The next payload which has to be included is the identification of the group to be joined (IDg). Optionally the GM can send an list of supported algorithms for the securing of the group communication, the server identity it requests the group from (IDr) or request a certificate from the GCKS. The IDr is needed if a physical entity contains several group managers handling different groups. When a SAg was included in the request it is evaluated if it matches requirements of the group. If no consensus is found the connection is rejected. The GCKS can use the combination of authentication payload, IDi and optional certificates to validate the identity of the GM. After this process is successfully completed the authorization process begins. The identification information contained in the IDg is used to check if the requested group is managed by the GCKS. If this is the case the IDi is compared to the allowed members for this group. On successful authorization the GCKS prepares the response in the GSA_AUTH exchange. The response contains the GCKS's identification, an authentication payload, an optional certificate and the group related payloads. The group related payloads consist of the GSA payload containing the algorithms to be used during group communication and the key download payload containing the keys to be used for the algorithms set in the GSA payload. All payloads except the header of the response are encrypted the same way the request was. The group member can then decrypt the main payload, authenticate the GCKS and then use the provided data in the GSA and KD payloads to create the necessary SA for the group communication. If it has not previously provided a SAg and does not support one of the chosen algorithms it has to notify the GCKS and then delete the SA corresponding to the GCKS. Otherwise it stores the created SA for the group communication in its SA database.

5.3 Databases

To manage groups and their keys several types of data are needed. To manage this data it is organized and kept in databases. They are closely associated with each other since all information combined is needed for the successful establishing of a secured connection. The Security Policy Database (SPD) contains the information needed to determine which connections are to be secured and how. Part of this information is stored in the traffic selectors. Traffic selectors contain the host addresses and port ranges and the transport layer protocol which are to be used for possible IPsec connections to this partner. This means that for every partner several traffic selectors can exist to distinguish by transport layer protocol and ports which allows the usage of differently secured connections. But also several connections can be managed by one traffic selector if a range of host addresses is given. The database also includes information on policies which are in effect for the described connection. This data incorporates things like which IPsec protocol is used in which transport mode and how long policies are in effect before they need to be refreshed. For group communication a minimum of two entries are needed, one for the registration process and one for the planned group communication. The second database is the Security Association Database (SAD). Here the SAs of the system are stored after they are negotiated. In the case of group management several SAs are needed. One is created during the registration which can be used for registration/deregistration exchanges. This is also called the management SA. It can also be used for the distribution of a new KEK should this be needed. Otherwise a rekeying SA can be created for secure transfer of new group keying information. The Group

Security Association (GSA) is also stored by each group member.

5.4 Group Management

Group management consists of the management of several factors. It incorporates the responsibilities of authenticating requesting parties and determining their identity. This can be done by the use of certificates issued by a trusted certification authority. This system is commonly used in cloud and grid computing but might have limited application in networks where part of the devices is constrained. A simpler possibility is the combination of an identification and pre-shared secrets, which however is not as secure. G-IKEv2 can use the same authentication mechanisms as IKEv2. In addition to the authentication a group manager also has to take care of the authorization. A simple possibility for this is the usage of a list associating access to the group with a id. More complex systems could include a database linking identifications to roles which themselves are associated with rights. In the scope of G-IKEv2 this would be either allowed to send and receive or not since a symmetric encryption is intended. In an asymmetric system this can be changed to allow the differentiation between senders and receivers. The current system is expandable in this direction should the need arise. The group manager has to keep track of the joining and leaving of group members and the associated roles. This can range from simple group access rights to a more complex system including sender and listener distinctions.

5.5 Strongswan

Since G-IKEv2 strongly resembles IKEv2 implementations of IKEv2 were shortly analyzed. Two major open-source implementations presented themselves as possible candidates for adaptation. Strongswan and Libreswan which both originate from a common parent project (FreeS/Wan) which was stopped, but are developed and maintained separately. Both would be feasible for the integration of a G-IKEv2 group management server. However Strongswan has undergone a re-implementation of IKEv2 which was thought to increase code homogeneity. Furthermore its primary focus was the implementation of IKEv2 whereas IKEv1 compatibility was added later on. Strongswan is used for the creation of VPN tunnels or IPsec secured connections. It can perform as either initiator or responder and offers a wide variety of security algorithms. It also is host to a number of plugins which further increases the available encryption algorithms should the necessary libraries be installed. Strongswan provides abstraction of the underlying kernel functions regarding the handling of UDP packets. This removes the need to implement an interface to the kernels network stack. The negotiated SAs and security policies are installed on the kernel which then manages the actual securing of network traffic. Due to the overlap of the protocols several methods can be used as provided, some have to be adapted and only the ones directly relating to group key management needed to be integrated.

5.6 Integration in Strongswan

The following section shows the structures needed for the implementation of G-IKEv2 and the processes and structures which are provided by Strongswan to create an overview over the tasks needed for the group key management.

5.6.1 Group Management and Databases

The first thing which needs to be introduced into Strongswan is a group manager structure to provide a database to store group related information. The group manager itself consists of an internal structure containing a list of functions which are used for the management of group and member structures. It provides an interface for the rest of Strongswan to access the group manager entity. Additionally there is a structure containing the group entries for managed groups. Each group entry contains a list of allowed members to manage access rights for this group. It also contains the policies which are the basis for the GSA payload needed in the phase 2 exchange of the protocol. The information stored here could also be stored in the provided SA database as it is a security association, but it was kept separate for testing purposes and could be merged in the future. A group entry also includes the keys needed for the corresponding GSA. Additionally each group entry is connected to a separate key generator to allow independent key generation between the groups. Upon initialization of Strongswan the group manager also has to be created and primed with information. This has to be done by the use of configuration files. The IKE daemon of Strongswan parses the configurations to fill its internal databases containing the traffic selectors, configurations and SAs. When parsing the configurations the daemon was expanded to initate a call to the group manager. The configuration entry is then analyzed if it is group related and if this is the case the information which is needed for the group database is extracted and stored. This includes the policies and the multicast address associated with the group. In the case of group member configurations the member is added to the corresponding group. The group member entries should also be linked to authentication methods and required data in the form of certificates or pre-shared keys. These should be stored separately with restricted access to prevent tampering with this critical information. In the case of G-IKEv2 in combination with Strongswan the connection data is stored in the ipsec.conf whereas secrets are stored in the ipsec.secrets file. The latter is only accessible with root privileges.

5.6.2 Architecture

Strongswan creates a task manager for each IKE-SA to handle tasks associated with the message exchanges. The concept of this is shown in 5.4. Depending on the role the own device takes on in the connection either as initiator or responder the appropriate methods are used. In the case of this thesis it acts as responder to the requests sent by the prospective group member, which is also the case which will be showcased here. Upon receipt of a message the major protocol version is read and the corresponding task manager started and associated with an IKE-SA. If the task manager has no scheduled task, as is to be expected after being freshly initialized, the exchange type of the message is analyzed to determine which tasks have to be scheduled. In the case of an IKE-SA-INIT request all the tasks needed in both phases of the two-phase exchange are scheduled. The GSA-AUTH was also added at this point since it replaces the functionality of the IKE-AUTH at this point. Due to the exclusive nature of these tasks a mechanisms was implemented to determine at the beginning of phase 2 which task to remove from the list. After the queuing is finished the list is iterated through two times. On the first run the process functions of the tasks are called to process the incoming request. On the second run the payloads needed for the response are built. Tasks remain on the list as long as they do not finish successfully. If all tasks scheduled have finished successfully, the response message is generated from the payloads

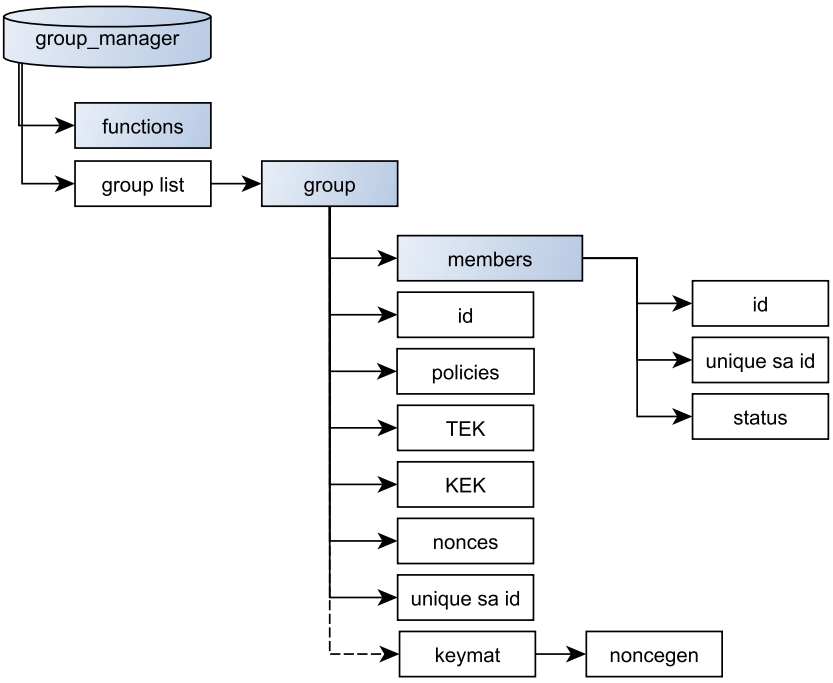

Figure 5.3: Structure of the internal group manager storing information related to the groups and their management. Blue boxes represent structures.

and subsequently sent.

5.6.3 Messages

The message exchange used in G-IKEv2 is shown in 5.5 and the steps on the GCKS's side are elaborated.

The first exchange which has to be working is the INIT phase. Here the secret used for the encryption of the next exchange is generated and the first step for the authentication prepared. 5.6 shows the tasks which need to be performed by the group key manager. The configuration which is associated with the current connection is loaded to gain access to the policies which are needed for the initial exchange. More precisely, the Diffie-Hellman group to use, what kind of authentication is requested and which algorithms are acceptable for the derivation of the key. The responder then has to generate a nonce which is to be used later on.

Then the SA payload is parsed to analyze the proposed algorithms and choose the best fitting to the own requirements which are store in the IKE-CFG. After selecting a proposal the key exchange (KE) payload is read and the Diffie-Hellman group extracted. After this the nonce payload is parsed and together with the own previously generated nonce the Diffie-Hellman secret is generated. Should any of the steps fail the communication is aborted and the initiated SA is deleted.

Prior to handling the GSA-AUTH the exchange type has to be checked if the requested connection is a GSA-AUTH or an IKE-AUTH. This could theoretically be determined by

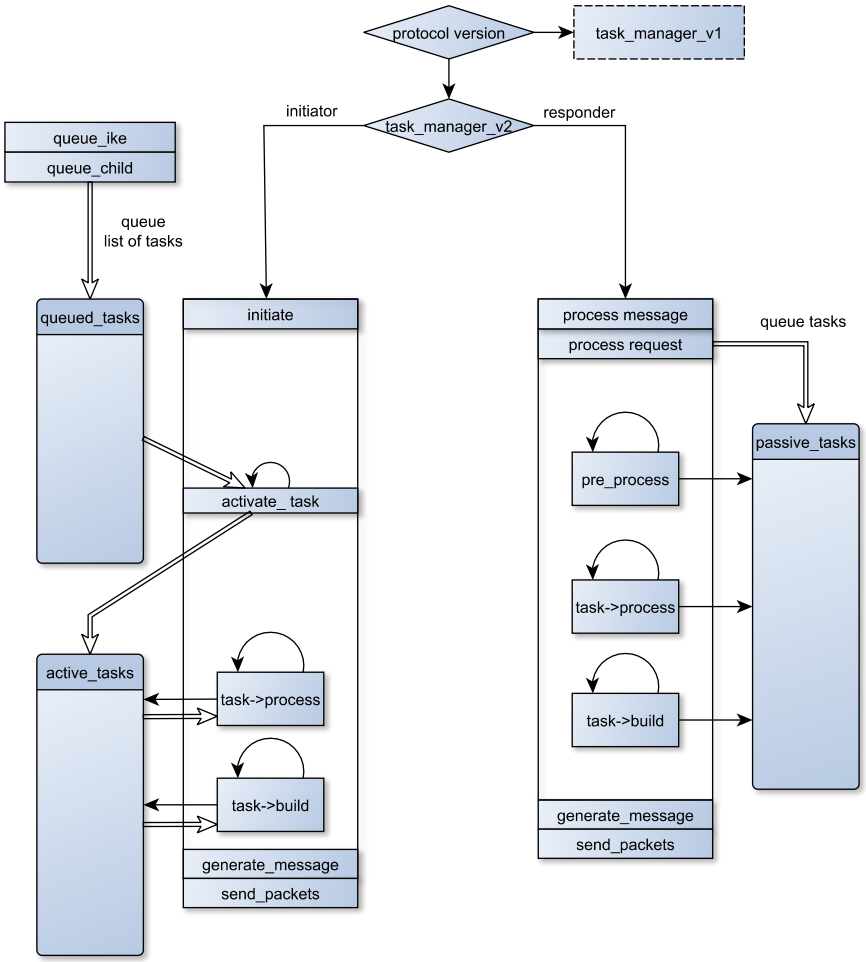

Figure 5.4: Structure of the internal task manager handling tasks for processing and creation of messages.

the port used for communication, but since this can be changed by a user it is not reliable. This cannot be determined earlier since IKEv2 and G-IKEv2 are identical up to that point. To distinguish between the different protocols both tasks are scheduled and according to the exchange type one of the processes is removed upon reaching the processing phase. This allows the usage of the system provided by Strongswan without having to unnecessarily duplicate code. 5.7 shows the process of handling a GSA-AUTH message after decrypting. The payloads extracted in this process are stored in an internal message object. New mechanisms needed solely for G-IKEv2 are highlighted in yellow. Using the sender information the corresponding IKE-SA is loaded to be accessible for updates during the process. After this the payloads are analyzed in the order presented starting with the initiators ID (IDi). If no IDi payload is available the process aborts and the SA is deleted. Otherwise the identification is extracted from the IDi payload and used to request the corresponding configurations from the internal databases. This includes the authentication method and policies which are needed later on. The IKE-SA is updated to now contain the initiators identification.

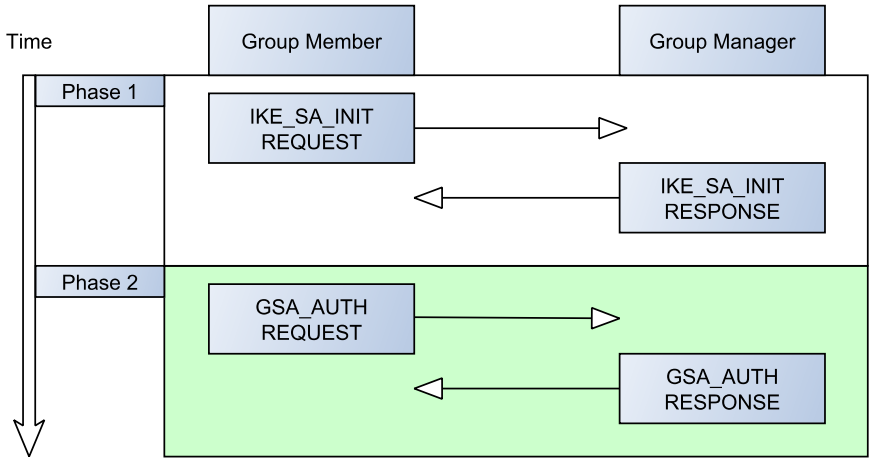

Figure 5.5: Initial registration process including authentication

Then the presence of a group identification payload is checked. If no group ID (IDg) is contained in the request the request is rejected and the SA then deleted. After successfully determining the presence of an IDg the AUTH payload is interpreted and the authentication process started. The authentication is performed either by usage of a provided certificate or by testing knowledge of a pre-shared secret. The authentication process is already established in Strongswan as it is an integral part of the IKEv2 protocol. After this part an important part of the G-IKEv2 protocol is performed, the authorization and registration of the initiator. The next steps query the group manager using the extracted group and initiator identification. The presence of the requested group is verified and the access rights of the initiator determined. If the results are positive the GM is registered in the group manager and the IKE-SA is referenced within the group structure. This is the last task directly involved in parsing the GSA-AUTH message. The other tasks which are included in Strongswan take care of managing specific IKEv2 functions not applicable to G-IKEv2 or are used to manage certificate handling.

After the processing tasks are finished the task list is used to call the build function of each task. On this second iteration of the GSA-AUTH task the responder identification is used to create the IDr payload. The payload is stored in the internal message construct. Then the AUTH payload for the response and a possible certificate payload are generated in the following tasks.

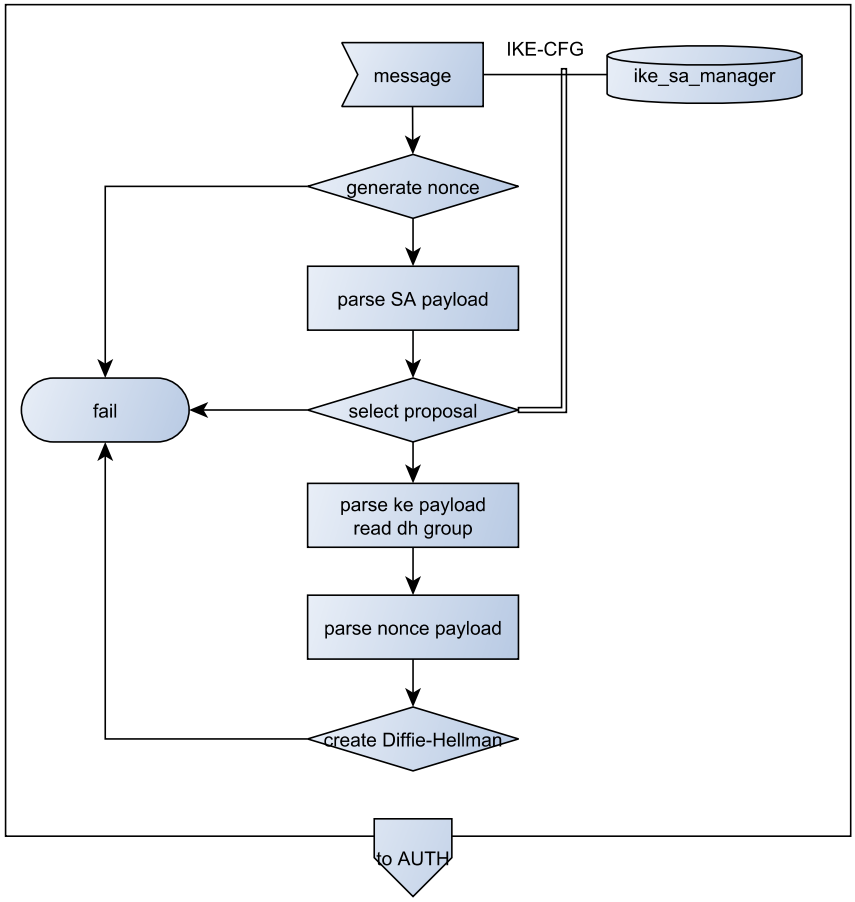

Figure 5.6: Processing of the IKE-SA-INIT request by the group manager as implemented by Strongswan

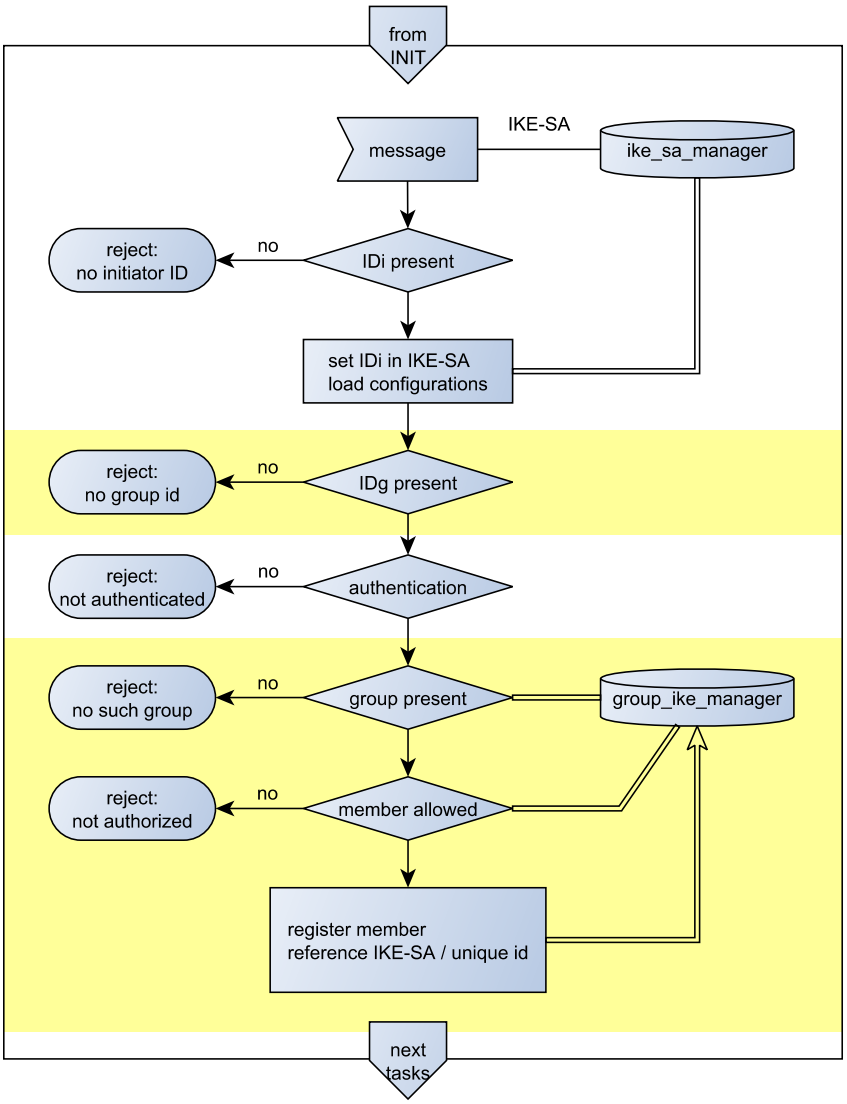

Figure 5.7: Processes needed for the handling of the second exchange. Yellow background shows the parts that need to be included into Strongswan

6 Implementation

This section describes the implementation of the G-IKEv2 draft version 11 into the IPsec suite Strongswan in the order operations have to be performed by the GCKS. Strongswan was chosen since it implements the majority of IKEv2 functionality for UNIX based systems. It is provided as open source and allows users to extend the functionality. Furthermore the implementation of group management for neither GDOI or G-IKEv2 was on the agenda for the main supporters of the project. This presented an opportunity to use the existing framework and include the basics of a group key management. Strongswan is written in the programming language C to provide sufficient low level control. However the coding style is very object oriented and several custom made constructs are used to mimic mainly object oriented programming languages. This provides interchangeability for functions and mechanisms but also makes tracking of call chains more complex. This is very visible with the implemented interfaces which are present in the defined payload structures to provide common functionality with a simple call. This coding style was also used for the newly implemented structures to seamlessly integrate. It also was required by the contributor guidelines of Strongswan. The implementation is integrated directly into the current functionality and not written as a plugin due to the strong interconnectedness to IKEv2. This required the adaptation of some IKEv2 methods to allow the distinction to G-IKEv2 since the protocols share many basic functions. Such alterations were kept to a minimum to not unnecessarily increase the maintenance effort for IKEv2 features. The payload structures were integrated as separate "classes" in the example of the already present structures. The code structure resembles the logical structure proposed in the IKEv2 RFC and in the case of the new code in the structure of the draft.

The provided transform_attribute functionality was expanded to also incorporate the TEK, KEK, and GAP attributes since they share the same basic structure. The GSA structures were newly implemented and allow the simple incorporation of not yet implemented features. The following section showcases selected elements of the daemon startup and running process which are needed for the G-IKEv2 protocol and the changes and additions which were needed to include group functionality.

```
struct private_group_ike_manager_t{
    group_ike_manager_t public;
    linked_list_t *groups;
};
```

Figure 6.1: Group manager internal structure

6.1 Group Manager Structure

The group manager itself consists of an internal structure containing a public interface and a linked list of group entries as shown in 6.1. This is a simple structure but can easily be replaced by a more complex management structure since functions were written in a way that they can be replaced without having to change code outside of the group manager. Each group entry contains a list of allowed members to manage access rights for this group. The knowledge of the authentication secret and the entry in this list serve as authorization to join this group. Currently a second list of members is kept to manage group members which successfully registered with the key manager. This will be replaced with a flag determining the state of the members to decrease memory consumption since originally planned differentiation is no longer needed in the current setup.

The public interface provides a set of functions which need to be accessed from other functions to separate them from internal functions of the group manager. The functions contained provide capabilities for the adding and updating of group and group member entries and the retrieval of the group SAs. The way this is built allows the casting of the public type to the private type for use within the "class" since the public interface contains only fixed value sizes due to them being either simple data types or pointers to functions.

The group structures each contain their own key generator with access to a wide array of encryption mechanisms to allow independent key generation. The keys are currently stored within the structure itself which has to be changed at some point to provide more security. As rekeying is not yet implemented the joining of a new group member does not prompt the server to distribute new keys. In the current setup the GM has to reregister when the key looses validity.

6.2 Startup

Upon startup the daemon initializes its managers which take care of the internal databases and the communication with the databases kept in the kernel. This function was expanded to also include the startup of the group manager to create a single instance of it. This is needed since the functions provided by the group manager are used at different positions during the runtime of the daemon and coherency is needed. A call to the destroy function was also added to properly destroy the manager on shutdown and prevent memory being left behind.

After creation of the managers the parent process of the IPsec daemon (charon daemon) is responsible for the parsing of the available configuration files and proceeds to send them to the charon daemon to process. Upon receiving these messages the peer and ike configuration structures in Strongswan are built. They contain the addresses and ports of the connection partners and which authentication methods are to be used. At this point a call to the group manager was added to determine if the configuration is related to G-IKEv2. Example entries for group connections are provided in 6.2.

The manager parses the name strings of the current entry to determine if a group or group member configuration has been created. If a G-IKEv2 related entry was found the group id is extracted from the name string. The internal database is then queried if an entry with this id already exists, as only unique entries are accepted. In the case of multiple entries only the first is accepted. If the entry corresponds to a group member and not a group the entry

```
# GROUP-IKE GROUP 1 DEFINITION
conn GIKE-G000010
     right=239.2.2.2
     authby=psk
     auto=add

# GROUP-IKE GROUP 1 MEMBER DEFINITIONS
conn GIKE-G000010MEMBER000001
#Own address
     left=192.168.178.161
     leftsubnet=192.168.178.0/24
#Client address
     right=192.168.178.120
     rightsubnet=192.168.178.0/24
     authby=psk
     auto=add
```

Figure 6.2: Group specific entries

is only accepted if the associated group exists and the member id is not already present. If the configuration corresponds to a new entry the appropriate structure is created and added in the group manager. Entries contain links to the associated configuration structures in the databases which are also created in this step. The identifications of the group and the group members are created from the provided "right" parameter. If the configuration does not contain any of the specifiers for a G-IKEv2 related entry the group manager ignores the entry. After this initialization the group manager contains a list of groups which it manages.

6.3 Registration exchange

Strongswan has been configured to accept IKEv2 messages on port 848 instead of 500 without handling them differently. When a message is received the header is analyzed to check the exchange type. This determines the way Strongswan handles the message. The message handling is done by a task manager scheduling the appropriate tasks for the exchange type received. In the case of G-IKEv2 the task manager for IKEv2 connections is used since they both are based on version 2 of the protocol.

If the exchange type is known, the payloads are copied into an internal message structure. All payloads of an exchange are added to or read from this structure. This is used to store received data which will be used in a later stage of the process. When receiving a IKE-SA-INIT request the tasks for an IKEv2 two phase exchange are scheduled in the task_manager2. All tasks needed for the initialization of an IKE_SA are scheduled. This is done so that tasks responsible for the second phase can extract and keep information from the INIT exchange. They are kept in memory until they finish their task successfully or fail, in which case the tasks are removed. If they are called in the wrong phase they have determined actions to be performed and return into a waiting state. Data kept this way from the INIT exchange include the nonces for example.

```
//Return SUCCESS to remove task from scheduled tasks, since it is not needed
    if(message->get_exchange_type(message) == GSA_AUTH){
        DBG1(DBG_IKE, "arrived in IKE AUTH for GSA!");
        return SUCCESS;
    }
```

Figure 6.3: Switch for the exclusion of the IKE task if the message is a GSA-Auth

```
idg_payload = (id_payload_t*)message->get_payload(message, PLV2_IDG);
idg = idg_payload->get_identification(idg_payload);
```

Figure 6.4: Extraction of the identification from the IDg payload

The task list is iterated through once to process the received message and afterwards the build functions of each task are called to create the payloads for the response. Since the GSA_AUTH and the IKE_AUTH share the same spot in the task order they needed to exclude each other. This is done by including checks for the exchange type of the message. Upon receiving a GSA_AUTH the IKE_AUTH task is set to SUCCESS so it gets destroyed and vice versa.

During the INIT phase the GSA_AUTH tasks retrieves the data it needs during the authentication phase. The details of the INIT phase are omitted here since the process in Strongswan was not changed from the IKEv2 implementation which follows the RFC for IKEv2. The INIT phase is handled by the ike_init task. The incoming payloads are processed according to the type specified in each payload header.

The GSA-AUTH task uses many functions which are the same as in the IKE-AUTH task since the main functions are still needed in G-IKEv2. It differs in the payloads it processes and builds. During the processing stage it first retrieves the client identification from the IDi payload and uses this to select the correct authentication configuration. Afterwards the IDg payload is retrieved which can contain different types of identification information like IP4 or IP6 addresses, hostnames or simple numbers. The identification is extracted from the payload and used for the lookup of managed groups 6.4. After the successful detection of a registered group the information of the group is searched for the provided identification of the initiator. This serves as the authorization step, as only members included in the allowed_members list are able to join. On successful authorization the authentication of the included payloads is performed. This deviates from the order proposed in the G-IKEv2 draft since the lookup of group and member are faster than the authentication which does not have to be performed if the initiator is not allowed to join. If the authentication fails the started IKE-SA is deleted and the GM would have to restart the G-IKEv2 exchange.

Only after authorization and authentication are passed successfully, the GM is added to the list of currently registered members for the requested group in the internal database and the AUTH exchange continues. The next tasks scheduled in the task manager are of no deeper interest for the implementation of the group manager since they manage the processing of certificates and the contained data is made available to the other tasks. This is assumed to be working as intended for the purposes of this thesis. The next tasks which was adapted was the tasks used for the creation of the child SA. Since no child SA is created

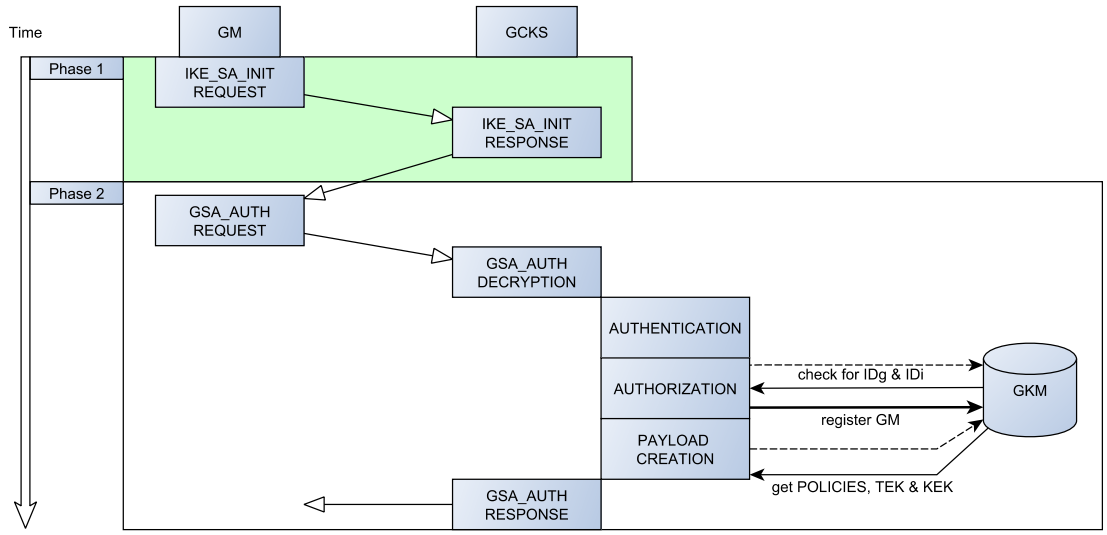

Figure 6.5: Registration process in G-IKEv2

in the G-IKEv2 protocol the parts needed for this are skipped with the inclusion of a switch. The GCKS then starts building the last payloads needed for the AUTH response. This is the GSA payload which contains the set of policies which are applicable in the group i.e. the message authentication, encryption and integrity algorithms as well as key lifetimes and other management parameters. The GSA payload also contains the security parameter indexes used to authenticate a subsequent KD payload. This KD payload contains the keys corresponding to the algorithms chosen in the GSA payload. In the most simple case these two payloads are encased in the same message. The policies are to be retrieved from the internal configurations referenced in the group manager and stored in the Strongswan specific peer configuration database. For testing purposes these values are currently manually set. The keys are generated at the creation of a group and read from the group structure when requested in the exchange.

6.4 Building the message

Strongswan keeps a set of rules for each exchange type for each direction to determine which payloads are to be included in which order 6.6. There also is a structure which includes information as to how many of each payload are required and how many are allowed as maximum. This also includes information as to which payloads need to be included in the encrypted payload.

This determines which payloads need to be encrypted and how many of them are required and which are optional. After the tasks for the creation of the payloads are finished the response is being built. For this the rules associated with the exchange type are used to order the payloads correctly. The payload sizes are read from the payloads and substructures to allocate the correct amount of memory and then the payloads are recursively copied to the new memory area. This was adapted to remove a previously existing limitation pertaining the usage of lists in the encoding rules for payloads. The pointer address used for the

```
static payload_order_t g_ike_auth_r_order[] = {
/* payload type                    notify type */
    {PLV2_ID_RESPONDER,             0},
    {PLV2_CERTIFICATE,              0},
    {PLV2_AUTH,                     0},
    {PLV2_GSA,                      0},
    {PLV2_KD,                       0},
    {PLV2_FRAGMENT,                 0},
};
```

Figure 6.6: Structure to determine the payload order for the GSA-AUTH response as implemented in Strongswan.

identification was overwritten in the case of lists. This has been changed to temporarily store the address to reset it after the call used for the handling of a list is finished. Furthermore correct handling for transform attribute types outside of lists was added. Payloads that need encryption are then encrypted and the message sent via the use of kernel functions.

7 Evaluation

The first part correlates the capabilites of G-IKEv2 with the requirements listed in chapter 3. Data replay protection is guaranteed by the use of SPIs which are linearly increasing which only get reset on certain rekeying operations. In the case of rekeying messages only message IDs higher than those previously received are handled.

As the group key manager authenticates and authorizes each group member upon registration and the ability to securely distribute policies and keys for authentication, integrity verification and encryption of data with the GSA and KD payloads the requirements for group confidentiality, authentication and data integrity are given.

As group members can propose supported algorithms the group key manager can pick the strongest algorithm by all group members and subsequently initiate a rekeying to enforce its use in the group.

The creation of a GSA is inherent in the protocol as was previously mentioned. A rekeying protocol is integrated in the G-IKEv2 protocol which can also be used to provide forward and backward secrecy. If systems like the logical key hierarchy are integrated the scalability of groups is given as G-IKEv2 can then perform many of the required group key management functions using multicast. The information distributed in the GSA and KD payloads is independent of the used protocol. The usage and interpretation of the distributed GSA is up to the client and the information contained in the payloads.

Implementation test To test the implementation of the GCKS within Strongswan the modified version was compiled and run on systems with differing Hardware. The tests were performed against a virtual machine running a CISCO CSR1000v router implementing an earlier version of the G-IKEv2 draft. The virtual machine was created and run with the VirtualBox from Oracle and provided with 6.5 GB of RAM to ensure a surplus of memory compared with the minimum requirements. A second round of tests was run against a constrained device running the minimal client developed by [FGHK17] which is based on a more recent version of the draft. It was tested if the connection and the key exchange were able to successfully finish. The modified Strongswan acting as server was compiled and executed on three different systems with decreasing resources available. Table 7.1 shows the specifications of the used systems.

For the first tests the router was configured to run as a client which should join a specified

Type	OS	RAM	Clock Speed (cores)
Virtual Machine	Ubuntu 17.10 (64 Bit)	11 GB	4 GHz (4)
Notebook	Ubuntu 16.04 (32 Bit)	2.9 GB	2.2 GHz (2)
Raspberry Pi 3 Model B	Raspbian	1GB	1.2 GHz (4)

Figure 7.1: Specifications of the hardware used to run the Group Key Manager

group. The CISCO was provided with the IP-address of the server and the group identification it should join. The authentication was configured to use a pre-shared key which was also configured on both sides. The server was provided with a set of possible groups and authorized group members. The configuration for both server and client are provided in Appendix B. During the initial exchange, the IKE-SA-INIT, both devices successfully selected common security algorithms, exchanged nonces and finished a Diffie-Hellman exchange. In the next exchange the negotiated key was used for the encryption of the payloads which could be decrypted showing that the previous exchange was successful. During the GSA-AUTH exchange the GCKS also parsed the incoming request correctly, identifying the request as GSA-AUTH and removing the IKE-AUTH task from the internal scheduled list of tasks. It then succeeded in finding the corresponding entry in the group managers data structure, thereby authorizing the GM. The construction of the payloads finished but upon receipt of the message it was detected that the message length did not match. This problem persisted even after rechecking the payload length calculations were corrected to fit latest changes in the structures. The problem was found in the way the message generation worked. The function creating the unencoded payloads, in correct order according to the internally stored rules, did not account for more than one list of substructures to appear in encoding rules. The list was designed to be the last entry in encoding rules which is not the case with the encoding rules in G-IKEv2. This lead to problems with the recursive nature of the message generation function. The function was adapted by storing the structure pointer before the call to the next recursion and restoring it afterwards. To ensure the correct generation of the message the debug output for the byte array generated was compared to the expected data when using the rules and input data by hand. The message was now correctly built, encrypted and subsequently sent. The GM identified the message correctly and successfully decrypted the main payload. However during the parsing of the GSA-payload the data interpretation of the client did not match the sent data, leading to incorrect values and an error in handling the next payload structure. Upon examination of the received bytes it was found that the parsing rules exhibited a shift of 4 Bytes prior to the SPI field of the payload header. To work around this incorrect implementation on the client's side for testing purposes the sent data was adapted to include 4 empty bytes in the payload header before the SPI. On subsequent tests the parsing of the header continued correctly until the client aborted the parsing of the GSA payload at different positions. This could be partly influenced by the order in which the substructures were included in the payload leading to the assumption that no special entry in the substructures lead to this error. The order in which the single substructures are ordered within the GSA payload should not influence parsing as each substructure carries the information needed for the correct parsing as is detailed in the draft by Yeung et al.. The CISCO router aborted parsing without returning any error message and simply resent a request. The 4 Byte shift in parsing the GSA header possibly originates from the different version of the draft used in the development of the router and version 11 which was used for this thesis. The adaptation integrated to obtain compatibility with the router was removed again since it does not adhere to the specifications set in version 11. Due to the mismatch in implemented protocol versions further tests with the router were not performed. The next tests were performed against an Arduino M0 Pro hardware running the minimal G-IKEv2 client built by [FGHK17]. This client is based on the same draft version as this thesis therefore preventing the previously arisen problems. After adapting the configuration to recognize the new client as valid group member the server was started and the client tried connecting. The IKE-SA-INIT failed due to an unsupported elliptic curve algorithm

being used by the client for the Diffie-Hellman exchange. This was fixed by installing the openssl library and configuring Strongswan to use its ssl plugin. After this simple fix the IKE-SA-INIT and GSA-AUTH exchange finished successfully without further adaptation.

8 Conclusion and Future Work

The G-IKEv2 protocol proposed in [WNS17] and implemented in this thesis provides simple and short registration and key distribution processes. It manages the authentication process which is performed either by usage of certificates or the use of a pre-shared key, both functions which are already provided by Strongswan. The Extensible Authentication Protocol (EAP) [ABV+04] is also implemented in Strongswan allowing its use should it be required. G-IKEv2 also requires the GKM to authorize prospective group members to restrict access and create a group containing only trusted members. This is very important since in a multicast architecture it is not possible to authenticate the senders identity without additional methods. By managing the access and distributing the relevant policies a group based authentication is possible. The framework provided by Strongswan provides a wide array of algorithms for encryption, authentication and integrity verification which can be used in the implementation of G-IKEv2. For algorithms not directly supported by Strongswan several plugins allow the addition of interfaces to further algorithms. Due to the plugin system Strongswan can easily be adapted to fit future needs. The authentication system of Strongswan also allows the usage of external services like Kerberos.

Even though G-IKEv2 provides a more secure system than its predecessor it is more lightweight. The predecessor GDOI used more messages even when using an aggressive mode which reduced the amount of message needed for a successful key exchange. In G-IKEv2 the authentication exchange was combined with the key request and key distribution steps to reduce the messages needed to four. Systems with a similar group key management require more message to be sent or depend on a pre-existing secure connection. Also the protocol is not as overloaded as is the case with the group variant of DTLS. Here the TLS protocol had to be expanded once to support UDP as underlying protocol and then again prepended with an additional header to distinguish between logical communication channels. This also leads to the addition of further exchanges to fulfill some of the prerequisites.

While providing a strong framework, Strongswan has low requirements concerning the hardware. The inclusion of the group key management should only have a very light impact on requirements. The main impact on memory is the management of the group structures and possibly the logical key hierarchy when it is implemented. The group management can probably be integrated more closely into the available structures which would in turn mean that the policies for each group entry are similar to a standard entry in the configurations database. This would leave the database of group identities, the corresponding lists of member identities, the links to the respective policy entries and the associated keys as the additional overhead compared to IKEv2. Strongswan including the group key management was able to run with a low footprint on a Raspberry Pi 3 running Raspbian whereas the implementation running on a Cisco router needed a minimum of 4GB of RAM to even be able to successfully boot. This is most likely to the different field of application of the router and Strongswan. The low requirements make the application in a IoT environment more feasible as it is only to be regarded as positive if the group key server also has a low power consumption albeit higher than the devices it probably manages. The implemented draft

version complies with the version used in the minimal client created by [FGHK17] thereby allowing the setup of working IoT networks.

Future work Due to the not yet implemented rekeying process the server requires group members to repeat the initial registration process to gain a new TEK. This will need to be changed in future version as the rekeying process is also necessary for backwards secrecy when performed upon entry of a new member. The usage of backwards secrecy could also prompt the addition of another role which can be used in the authorization as privileged members could be excluded from the need for rekeying upon group entry. The rekeying methods should also take into account that the groups managed may not based on multicast. This would change rekeying behavior and require the GCKS to also distribute a list of group members in combination with the TEK. This however would be an addition to the protocol and therefore require more preparation. Another function which has to be expanded is the interpretation of the configuration files. For this the configuration parsers need to include G-IKEv2 specific tags to better distinguish group related entries from standard IKEv2 entries and to provide more data in a readable format. Strongswan possesses an alternative configuration interface which can also be accessed using side channels which could, after adapting current processes to allow the dynamic changing of groups, memberships and policies, be used to provide addition and removal of groups and member authorizations during runtime. The authentication of group members at the server could also be managed with Kerberos and possibly be expanded to also allow the usage for the authorization process for groups outsourcing these tasks from the group key manager.

If possible, sender authentication mechanisms should be integrated or the policies for this created so that groups can then make use of them. This would further protect against group members going rogue and sending messages reserved for the GCKS. It would also allow for the designation of a backup GCKS to take over if the main GCKS fails, thereby providing a partial solution to the problems of centralized key management systems like G-IKEv2. As currently the registration in a group requires the knowledge of the groups identification and the address of the associated group key manager it would be good to integrate systems to allow the dynamic detection of available groups. The possibility of a request to acquire a list of groups the initiator is authorized to join or the option to join all of them should also be seen as a possible expansion of the current protocol.

Appendix A: Installation instruction

Tools needed for compiling and installation:

- git
- gcc 3.x or higher
- automake
- (autoconf)
- libtool
- pkg-config
- gettext
- perl
- python
- bison
- flex
- gperf
- libgmp3-dev for gmp library
- libssl-dev for ECP-256

Procedure to install the server

- clone git repository
  currently at https://gitlab.cip.ifi.lmu.de/engelbrechtw/StrongswanG-Ikev2.git
- execute ./autogen.sh
- execute ./configure -enable-openssl –with-charon-udp-port=848
- make
- sudo make install
- Edit configuration files as needed.
- sudo ipsec start

Appendix B: Strongswan Example Configurations

This chapter show the rules for the configuration file to allow its use for G-IKEv2 and example configurations.

1 ipsec.conf

This file contains the connection information and the algorithms which are accepted

For group entries the configuration should follow these rules for the naming:

GIKE-G000001

This designates the entry as a G-IKEv2 group entry for group 1. The number is internally used as group id and has to be unique. This entry has to preceed any member entries pertaining to this group.

GIKE-G000001MEMBER000001

This designates the entry as group member 1 of group 1. The number has to be unique.

The "right" entry in the group entry designates the address which is used for group communication.

Example configuration:

```
# ipsec.conf - strongSwan IPsec configuration file
#
conn %default
        keyingtries=1
        keyexchange=ikev2
      ike=aes128-sha256-ecp256,aes256-sha384-ecp384,aes128-sha256-modp2048
      esp=aes128gcm16-ecp256,aes256gcm16-ecp384,aes128-sha256-ecp256
# GROUP-IKE GROUP 1 DEFINITION
conn GIKE-G000001
 left=192.168.0.102
     right=239.2.2.2
     authby=psk
     auto=add
# GROUP-IKE GROUP 1 MEMBER DEFINITIONS
conn GIKE-G000001MEMBER000001
#Own address
     left=192.168.0.102
     leftsubnet=192.168.0.0/24
#Client address
     right=192.168.0.101
     rightsubnet=192.168.0.0/24
```

```
authby=psk
auto=add
```

The "ike=" and "esp=" can be set for each connection separately but need to be appended with an ! to stop Strongswan from appending the default values to the list. A list of possible parameters and values is to be found in the Strongswan wiki at https://wiki.strongswan.org/projects/strongswan/wiki/

2 strongswan.conf

This configuration file takes care of general Strongswan related options. In the example files these are the logging options and the plugin loading.

```
# strongswan.conf - strongSwan configuration file
#
# Refer to the strongswan.conf(5) manpage for details
#
# Configuration changes should be made in the included files

charon {
# BEGIN optional for logging
        filelog{
                /var/log/charon.log{
                #add timestamp prefix
                time_format = %b %e %T
                #prepend connection name, simplifies grepping
                ike_name = yes
                #overwrite existing files
                append = no
                #increase default loglevel for all daemon subsystems
                default = 4
                #flush each line to disk
                flush_line = yes
                }
        }
        stderr{
                ike = 4
                cfg=4
                knl = 3
        }

# END logging

        load_modular = yes
        plugins {
                include strongswan.d/charon/*.conf
        }
```

```
}

include strongswan.d/*.conf
```

3 ipsec.secrets

This file contains the secrets used for the configured connections. They can overlap as internal Strongswan routines choose the entry with the highest consensus. The structure is detailed in the Strongswan wiki.

```
192.168.0.101 : PSK "secretkey"
%any : PSK "notarealsecret"
```

Appendix C: CISCO Router configuration

Configuration for the CISCO CSV1000 Router as a G-IKEv2 client

```
 crypto ikev2 profile sm2
 description G-IKEv2
 match identity remote any
 authentication local pre-share key "Test"
 authentication remote pre-share key "Test"

 crypto gkm group sm2
 !group identity for IDg:
 identity number 1
 !Key Server
 server address ipv4 192.168.178.31
 client protocol gikev2 sm2
!
! The 1 in the following line represents the group-id to join
! which can also be an address
crypto map sm2 1 gdoi
 set group sm2
!
interface GigabitEthernet 1
 description Interface that will be encrypting/decrypting traffic
 ip pim sparse-mode
```

List of Figures

Bibliography

[ABV+04] Aboba, B. ; Blunk, L. ; Vollbrecht, J. ; Carlson, J. ; Levkowetz, H.: Extensible Authentication Protocol (EAP) / RFC Editor. Version: June 2004. http://www.rfc-editor.org/rfc/rfc3748.txt. RFC Editor, June 2004 (3748). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc3748.txt

[BCDL05] Baugher, M. ; Canetti, R. ; Dondeti, L. ; Lindholm, F.: Multicast Security (MSEC) Group Key Management Architecture / RFC Editor. Version: April 2005. http://www.rfc-editor.org/rfc/rfc4046.txt. RFC Editor, April 2005 (4046). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc4046.txt

[CSF+08] Cooper, D. ; Santesson, S. ; Farrell, S. ; Boeyen, S. ; Housley, R. ; Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile / RFC Editor. Version: May 2008. http://www.rfc-editor.org/rfc/rfc5280.txt. RFC Editor, May 2008 (5280). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc5280.txt

[DR08] Dierks, T. ; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2 / RFC Editor. Version: August 2008. http://www.rfc-editor.org/rfc/rfc5246.txt. RFC Editor, August 2008 (5246). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc5246.txt

[FGHK17] Felde, N. G. ; Guggemos, T. ; Heider, T. ; KranzlmÃ¼ller, D.: Secure group key distribution in constrained environments with IKEv2. In: *2017 IEEE Conference on Dependable and Secure Computing*, 2017, S. 384–391

[GCB03] Gutierrez, Jose A. ; Callaway, Edgar H. ; Barrett, Raymond: *IEEE 802.15.4 Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensor Networks*. New York, NY, USA : IEEE Standards Office, 2003. – ISBN 0738135577

[HC98] Harkins, Dan ; Carrel, Dave: The Internet Key Exchange (IKE) / RFC Editor. Version: November 1998. http://www.rfc-editor.org/rfc/rfc2409.txt. RFC Editor, November 1998 (2409). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc2409.txt

[IEE16] IEEE: IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (2016), Dec, S. 1–3534. http://dx.doi.org/10.1109/IEEESTD.2016.7786995. – DOI 10.1109/IEEESTD.2016.7786995

[IHA+14] Ishaq, Isam ; Hoebeke, Jeroen ; Abeele, Floris Van d. ; Rossey, Jen ; Moerman, Ingrid ; Demeester, Piet: Flexible Unicast-Based Group Communication for CoAP-Enabled Devices. In: *Sensors* 14 (2014), Nr. 6, 9833–9877. http://dx.doi.org/10.3390/s140609833. – DOI 10.3390/s140609833. – ISSN 1424–8220

[KHN+14] Kaufman, C. ; Hoffman, P. ; Nir, Y. ; Eronen, P. ; Kivinen, T.: Internet Key Exchange Protocol Version 2 (IKEv2) / RFC Editor. Version: October 2014. http://www.rfc-editor.org/rfc/rfc7296.txt. RFC Editor, October 2014 (79). – STD. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc7296.txt

[KPT00] Kim, Yongdae ; Perrig, Adrian ; Tsudik, Gene: Simple and Fault-tolerant Key Agreement for Dynamic Collaborative Groups. In: *Proceedings of the 7th ACM Conference on Computer and Communications Security*. New York, NY, USA : ACM, 2000 (CCS '00). – ISBN 1–58113–203–4, 235–244

[KS05] Kent, S. ; Seo, K.: Security Architecture for the Internet Protocol / RFC Editor. Version: December 2005. http://www.rfc-editor.org/rfc/rfc4301.txt. RFC Editor, December 2005 (4301). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc4301.txt

[Pos81] Postel, Jon: Transmission Control Protocol / RFC Editor. Version: September 1981. http://www.rfc-editor.org/rfc/rfc793.txt. RFC Editor, September 1981 (7). – STD. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc793.txt

[RD14] Rahman, A. ; Dijk, E.: Group Communication for the Constrained Application Protocol (CoAP) / RFC Editor. Version: October 2014. http://www.rfc-editor.org/rfc/rfc7390.txt. RFC Editor, October 2014 (7390). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc7390.txt

[Res99] Rescorla, Eric: Diffie-Hellman Key Agreement Method / RFC Editor. Version: June 1999. http://www.rfc-editor.org/rfc/rfc2631.txt. RFC Editor, June 1999 (2631). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc2631.txt

[RM12] Rescorla, E. ; Modadugu, N.: Datagram Transport Layer Security Version 1.2 / RFC Editor. Version: January 2012. http://www.rfc-editor.org/rfc/rfc6347.txt. RFC Editor, January 2012 (6347). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc6347.txt

[SHB14] Shelby, Z. ; Hartke, K. ; Bormann, C.: The Constrained Application Protocol (CoAP) / RFC Editor. Version: June 2014. http://www.rfc-editor.org/rfc/rfc7252.txt. RFC Editor, June 2014 (7252). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc7252.txt

[TNR17] Tiloca, Marco ; Nikitin, Kirill ; Raza, Shahid: Axiom: DTLS-Based Secure IoT Group Communication. In: *ACM Trans. Embed. Comput. Syst.* 16 (2017), April, Nr. 3, 66:1–66:29. http://dx.doi.org/10.1145/3047413. – DOI 10.1145/3047413. – ISSN 1539–9087

[WNS17] Weis, Brian ; Nir, Yoav ; Smyslov, Valery: Group Key Management using IKEv2 / IETF Secretariat. Version: March 2017. http://www.ietf.org/internet-drafts/draft-yeung-g-ikev2-11.txt. 2017 (draft-yeung-g-ikev2-11). – Internet-Draft. – http://www.ietf.org/internet-drafts/draft-yeung-g-ikev2-11.txt

[WRH11] Weis, B. ; Rowles, S. ; Hardjono, T.: The Group Domain of Interpretation / RFC Editor. Version: October 2011. http://www.rfc-editor.org/rfc/rfc6407.txt. RFC Editor, October 2011 (6407). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc6407.txt