Secure Group Key Distribution in Constrained Environments with IKEv2

Nils gentschen Felde, Tobias Guggemos, Tobias Heider, Dieter Kranzlmüller
MNM-Team, Ludwig-Maximilians-Universität München, Munich, Germany
Email: {felde, guggemos, heidert, kranzlmueller}@nm.ifi.lmu.de

***Abstract*—Group communication is an important means for communication in todays interconnected world, where multiple endpoints need exchange of data in the most efficient and concurrently secure way. The resulting complexity represents a substantial challenge, especially in the constrained environments introduced through the Internet-of-Things and sensor networks. An in-depth analysis of existing work shows that the problem of secure key distribution within groups requires novel approaches. Instead of designing yet another (group) key distribution scheme, this paper offers a minimal client based on the well-known Group Internet-Key-Exchange protocol G-IKEv2. The evaluation of this first open client with real-life observations and corresponding measurements proves its applicability for secure group key distribution and will serve as the basis for implementing group and identity management.**

***Keywords*-IoT; Multicast; Security; Constrained Networks; Key Management.**

I. Introduction

In the absence of energy or bandwidth constraints, today's (fixed line) networks tend to emulate multicasting by a multitude of unicasts reducing group management to simple functionality on top of point-to-point communication. Unfortunately, energy, bandwidth and compute power is crucial, especially in constrained environments and wireless sensor networks (WSN). Thus, multicasting is meant to become more important, as it heavily reduces network overhead [1] and both, the required for CPU power and energy consumption. As a consequence, the need for secure group management and secure group key distribution in constrained environments increases and is not yet solved appropriately – neither in standards, nor in the literature and not as a suitable implementation.

In [2], a definition of secure group communication in constrained environments is given. It is shown that both sender authentication and secure group management mostly address confidentiality during transport, but neither authentication nor secure key exchange or secure group management are considered. These points are addressed by the contributions of this paper:

1) Analysis of existing secure group management approaches and their suitability for constrained networks (Section II)
2) Description of *Group-IKEv2* (G-IKEv2 [3]), which is based on *Internet Key Exchange v2* (IKEv2 [4]), together with an implementation of a minimal G-IKEv2 client (Section III-A)
3) Implementation and evaluation of a G-IKEv2 client for the RIOT [5] operating system (Section III-B and IV)

The next section reviews works on secure group management and evaluates existing approaches and related work concerning their suitability for constrained environments. With the results of Section II, G-IKEv2 [3] is shown best applicable for secure group key distribution in constrained networks. Section III designs a minimal G-IKEv2 client and summarizes its implementation to be deployed onto minimally equipped embedded / constrained systems. Furthermore, Section IV evaluates the client both analytically and using real-life observations and measurements, before Section V summarizes and concludes the paper.

II. On Secure Group Management

Secure group communication is referred to the insurance of a multitude of security attributes, such as accountability, availability, reliability, confidentiality, integrity, authenticity and non-repudiation in a communicating group [2]. In order to enable secure group communication, management of the group itself is found obligatory. Nevertheless, in many scenarios group management is designed in a centralized manner, which is the most common form of group management (e. g. Kerberos, X.509-PKI, etc.). For highly distributed and constrained networks like the Internet of Things (IoT) or wireless sensor networks, traditional group management approaches are not applicable due to various reasons, e. g. CPU power constraints, energy restrictions, highly distributed and dynamics environment, etc.

In the following, Section II-A extracts requirements for securely managing groups in constrained environments. Section II-B then elaborates in more detail on related work in terms of secure distribution of cryptographic material (such as keys), before Section II-C concludes by choosing a suitable protocol to cope with constrained environments in the sense of this work.

A. Requirements

In order to achieve secure group management, certain (security) requirements have to be met. This section summarizes obligatory security features from RFC 3740 [6] – with regard to the field of constrained environments – and adds additional requirements to also grant the not yet

978-1-5090-5569-2/17/$31.00 ©2017 IEEE

addressed $1:1$-communication pattern as a special case of group communication from [2]. To summarize, the following security features have been found mandatory:

Identity Management: The management of IDs ensures to uniquely identify a member of the group members or devices requesting access to the group.

Authorization and Authentication Infrastructure (AAI): Authentication requires some form of key distribution, such as a public key infrastructure (PKI), while authorization enables the management of rights for the group.

Group Key Management (GKM): Among others, GKM enables secure access to relevant information, such as group keys in order to grant confidentiality, integrity as well as sender- and group-authentication.

Group Management (GM): Basic group management operations, such as *createGroup*, *joinGroup*, *leaveGroup* or *destroyGroup* are a minimal subset of non-optional functions.

Security: Groups have additional requirements on security, especially after group operations have been carried out. A member leaving the group may still have access to messages received during its membership, but not any new message (forward secrecy). New group members need access to future messages, but must not be able to read any message before they joined the group (backward secrecy).

B. Related Work

During the past decade, several research activities on key distribution in different areas have been carried out [7]. It started with the key distribution for mobile networks (e. g. UMTS, GPRS, etc.), followed by activities to share keys in wireless sensor networks, while considering constrained environments. Salgarelli et al. [8] provide a solution for a wireless key exchange and authentication protocol, which has been compared to current approaches such as the *Extensible Authentication Protocol* (EAP [9]) and *Transport Layer Security* (TLS [10]). Although this work focuses on wireless networks and mobility, it still requires 12 messages to distribute a pair of keys. Additionally, it is not designed for multicast key distribution, which is essential when talking about group communication.

Group key distribution itself has been studied in [11], which resulted in a couple of standardization activities (some of them discussed in Section II-B). Rafaeli et al. [11] survey a set of approaches for secure group key distribution (GKD). According to their analysis, there are three different types of GKDs: centralized, decentralized and distributed GKD protocols. Most of the protocols considered are rather mathematical schemes than networking protocols, but nevertheless some of them are included in actual group management protocols, such as the *Group Domain of Interpretation* (GDOI) [12]. Although all of these approaches specify the possibility of secure (group) key distribution (also in constrained networks), none of them had been properly implemented, distributed or evaluated against the requirements in constrained environments such as 1) highly resource constraint devices, 2) highly distributed and 3) globally connected.

In the following, recent activities with the goal of a DTLS-based multicast solution for low latency networks [13] are analyzed and the most commonly used protocol (GDOI) and its potential replacement G-IKEv2 are explained (more on G-IKEv2 can be found in section III). Additionally, the concept behind hierarchical and distributed GKMs and their applicability in constrained networks is detailed.

1) Centralized Group Key Management: Centralized key distribution is the most obvious way of managing group keys as it leaves the complexity and trust to a single system. Most of the commonly used protocols (DTLS, GDOI, Kerberos, etc.) are designed on top of centralized systems. However, the concept of centralization inherits some natural difficulties, such as weak scalability, especially when only one server manages a geographically distributed network. In general, the larger a group gets, the more complex becomes its management and the resulting operations. This makes many centralized group key management concepts not feasible for constrained environments.

2) Decentralized Group Key Management: Decentralized key distribution still sticks to the concept of a central server, but splits the group in administrative domains for both management operations and key exchanges. Additionally, it distributes the workload over more devices and thus eases the key calculations on the client side. On the other hand, the decentralized concept requires strong trust relationships, which is a showstopper for many use cases where administrative domains can change frequently and devices are potentially physically accessible by arbitrary persons. As exemplary standards, GDOI and G-IKEv2 (see below) are able to use decentralized schemes for group key derivation.

3) Distributed Group Key Management: Distributed concepts remove any kind of management server, thus, every member of the group holds the complete state of the group. This approach produces additional network load and compute operations for re-keying every time a group management operation is performed. Thus, these concepts work well in closed and ”stable“ environments such as wireless sensor networks, but they obviously do not scale to a large extent.

4) DTLS-based: The IETF internet draft on *Security for Low-Latency Group Communication* [13] describes secure multicasting by using DTLS in low latency networks. The focus is the distribution of symmetric keys, which are also used for authentication and thus limiting the achievable level of security. Nevertheless, the draft designs a system including a key distribution center and authorization server, but it does not consider issues of forward and backward secrecy

Table I
COMPARISON OF EXISTING GROUP KEY EXCHANGE MECHANISMS

Feature / Requirement	DTLS	GDOI	G-IKEv2	Decentralized	Distributed
Group Management					
↪ Join Unicast	✘	✔	✔	✔	✔
↪ Join Multicast	✔	✔	✔	✔	✔
↪ Leave Unicast	✘	(✔)[a]	(✔)[a]	✔	✔
↪ Leave Multicast	✘	(✔)[a]	(✔)[a]	✔	✔
Security					
↪ *in general*	✔	(✔)[b]	✔	✔	✔
↪ Forward Secrecy	✘	✔[c]	✔[c]	(✔)	✔
↪ Backward Secrecy	✘	✔[c]	✔[c]	✔	✔
Applicability					
↪ Link Local	✘	✘	✔	✘	✘
↪ Broadcast Domain	✘	✘	✔	✔	✔
↪ LAN	✔	✔	✔	✔	✔
↪ WAN	✔	✔	✔	(✔)	✘
Lightweight:					
↪ Implementation	✔	✘	✔	✘	✘
↪ Memory/Storage	✔	(✔)	✔	✘	✘
↪ Networking	(✔)	✘	✔	(✔)	✘
↪ Standardized for IoT	✔	✘	(✔)	✘	✘

legend: ✔ addressed by design (✔) partially addressed ✘ not addressed by design
[a] *Leave* is only supported by the GKM server [b] based on obsoleted IKEv1
[c] with logical key hierarchy (LKH)

and re-keying. Additionally, no group management operations are considered, despite *joinGroup*-operations. Due to the design choice to use DTLS, the applicability is limited to local and wide area networks, but excludes possibility to be used on lower ISO/OSI-Layers. On the other hand, with RFC 7925 [14] DTLS is adjusted to comply with the needs and constraints of IoT devices.

5) IPsec based (GDOI and G-IKEv2): With multicast usually being an IP specific use case, the choice of IPsec for security seems natural. GDOI [12] was the first standardized protocol for securing multicast. It is a centralized protocol, but with the possibility of using hierarchical key distributions mechanism such as logical key hierarchy (LKH). However, due to the network overhead during key exchanges (7 messages) and using the obsolete IKEv1 as the underlying protocol, a replacement is inevitable. That said, G-IKEv2 [3] is currently proposed as a new standard for group key management. Using IKEv2 reduces the number of messages during key exchange to only 4, making it easier adoptable for constrained devices. Additionally, with RFC 7815 IKEv2 is specified for the use in IoT [4] and recommended for the key exchange in IEEE 802.15.4 networks.

C. Summary and choice of a protocol

Table I shows the findings of this section and evaluates related work against four categories of requirements:

Group Management describes the ability to manage groups for multicast and unicast, both of which are considered subsets of $n:m$-communication patterns.

Security evaluates on the security of a given solution in general and adds a special focus on forward and backward secrecy.

Applicability considers if a solution can be used in link local networks (e. g. radio networks) or broadcast domains (e. g. in self organizing WSNs) – both being ISO/OSI layer II networks – or local area networks (LAN, environments requiring local routing) or wide area networks (WAN) where global routing is required.

Lightweight shows if a solution can be found suitable for constrained devices in terms of their limitations (complexity of the implementation, limited CPU power, memory / storage capacities, network resources). Additionally, it is assessed on the availability of a specification specific to constrained devices and environments.

To conclude, G-IKEv2 is suited best for the area of application of this work. With its underlying protocol IKEv2 already being adopted (RFC 7815 [4]) and used in constrained environments and its flexibility to be extended by concepts of decentralized and distributed group key distribution schemes, it represents a good choice.

III. A MINIMAL G-IKEV2 CLIENT

This section focuses on the design of a minimal G-IKEv2 client (Section III-A) and its prototypical implementation for a sensor operating systems such as RIOT [5] (see also Section III-B).

A. Design

The main idea to achieve minimalism for G-IKEv2 is borrowed from RFC 7815 on *Minimal IKEv2* [4]. To achieve a minimized message exchange, the number of exchanged packages was kept minimal by using a minimal and predetermined configuration of the client rather than using compression.

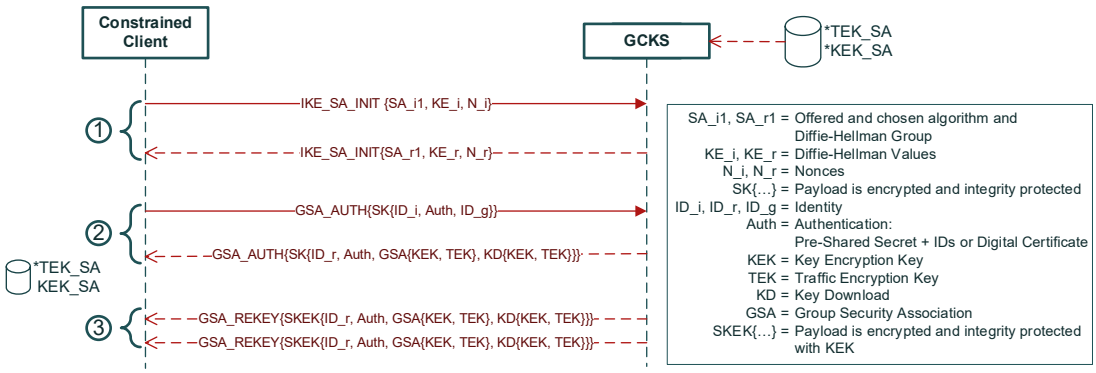


Figure 1. Simplified G-IKEv2 exchange for minimal implementations

Figure 1 illustrates a minimal G-IKEv2 key exchange and re-keying action.

Key Exchange. A G-IKEv2 key exchange can be divided into two phases:

① **Establishing an SA (IKE_SA_INIT).** The first two messages from the client to the *Group Communication Key Server* (GCKS) and back establish a *Security Association* (SA) and thus a secure channel between the client and the server.

② **Exchanging keys (GSA_AUTH).** Given the secured communication path, the client identifies and authenticates itself and in turn receives transport and key encryption keys (TEK and KEK) from the server.

③ **Re-Keying (GSA_REKEY).** Whenever a TEK loses validity (e. g. because it is outdated), a re-keying action is triggered by the server (GSA_REKEY), which is close to equal to the GSA_AUTH phase.

In the following, the different payloads (① – ③) are described in more detail and a special focus is put on how minimization could be achieved.

① *IKE_SA_INIT*

A Diffie-Hellman key exchange between the client (*initiator*, i) and the server (*responder*, r) is carried out during the IKE_SA_INIT phase in ①. In this case, the server holds the role of the group key management server. The IKE_SA_INIT is identical to the IKE_INIT of IKEv2 and does not include any group relevant information. Its only purpose is to establish a secure connection between the client and the server.

In the first step, the initiator sends a packet consisting of minimally the *Security Association* (SA), *Key Exchange* (KE) and a random number N_i – the so-called *Nonce*. Simply speaking, the SA payload contains a list of proposals for cipher suites to use for securing the following GSA_AUTH packet and the KE includes the initiator's Diffie-Hellman value and group. For a minimal implementation, exactly one proposal is sent, which the responder must accept. Upon acceptance, the GCKS responds (second packet as per Figure 1) by sending the same SA, his KE (including his Diffie-Hellman value) and Nonce (N_r). At this point, both the initiator and responder can derive the shared secret, which will be found the most time consuming operation during the key exchange (see Section IV).

② *GSA_AUTH*

Having successfully completed the IKE_SA_INIT phase results in a secured channel between the client and the server as a prerequisite for the GSA_AUTH phase in ②. The first GSA_AUTH message uses the same security properties as given by IKE_AUTH in IKEv2. The ID_i payload may include either a 4 Byte IPv4 address, or a 16 Byte IPv6 address or an arbitrary ID of variable length. The AUTH payload contains a hash of the IKE_SA_INIT request message, signed with either a digital signature or a pre-shared secret to eliminate possible man-in-the-middle attacks. Optionally, the initiator may also include an ID_g (Group-ID) payload that identifies the group he wishes to join.

It is important to note that unlike IKEv2's IKE_AUTH exchange, the initiator does not send any SA or traffic selector payloads in the request. Because group communication is targeted, the GCKS defines the configuration for transport security and thus, further negotiation is neither needed nor allowed or foreseen. For a minimal implementation, only one encryption and authentication function is implemented, which should be the one used for the group communication chosen by the server.

After having verified the initiator's identity and authenticity, the GCKS responds with its own ID, proves knowledge of the secret and additionally sends the GSA and KD payload containing the optional re-key SA, one or more group traffic encryption SAs and their corresponding keys.

The received SAs are stored in the *Security Association Database* (SAD), including the *Transport Encryption Key*s (TEK), for securing the group communication between the clients and *Key Encryption Key*s (KEK) for securing the re-keying of newly distributed TEKs. Please note, if forward and backward secrecy of the communication is required, the KEK can be generated by functions such as *Logical Key*

Hierarchy (LKH, see [11]). This requires the implementation of the *Download Type* payload, which is able to transport different schemes for re-keying, one of them being LKH.

③ *GSA_REKEY*

The GSA_REKEY message is the only message required for a re-key action and is initiated by the GCKS. It contains an AUTH payload to prove knowledge of the auth-secret that was previously shared in the KD payload, a new GSA payload and the new keys in a KD payload. The message itself is protected with the algorithms and keys that had been saved in the KEK SA, exchanged earlier during the GSA_AUTH response (see above). Please note, if forward secrecy is required, the KEK and TEK needs to be re-keyed within to separate GSA_REKEY messages, otherwise a left member could still have access to new TEKs.

The client is required to match the incoming GSA_REKEY packet to the correct re-key SA, validate the AUTH payload and update its SAD entries with the new attributes and keys. Although the message is typically sent using multicast (and thus does not require a response), certain re-keying schemes (e. g. LKH) require GSA_REKEY requests to be sent in unicast.

Table II
MEMORY NEED OF DIFFERENT CIPHERS

Cipher-Suite	**IKE_SA_INIT**	**GSA_AUTH**	**Key length (2 per SA)**
Fixed Size	88 B	98 B	—
Encryption:			
- AES128-CBC	—	+ 16 B + 15 B[a]	16 B
- AES256-CBC	—	+ 16 B + 15 B[a]	32 B
Integrity:			
- SHA1-96	—	+ 12 B	20 B
- SHA256-160	—	+ 16 B	32 B
- AES-XCBC-96	—	+ 12 B	16 B
Combined:			
- AES128-CCM	- 8 B[b]	+ 8 B + 8 B[a]	19 B
- AES256-CCM	- 8 B[b]	+ 8 B + 8 B[a]	35 B
- AES128-GCM	- 8 B[b]	+ 8 B + 8 B[a]	19 B
- AES256-GCM	- 8 B[b]	+ 8 B + 8 B[a]	35 B
PRF:			
- SHA1-96	—	+ 20 B	—
- SHA256-160	—	+ 32 B	—
DH Group:			
- ECP256	+ 64 B	—	—
- curve25519	+ 64 B	—	—
- 2048MODP	+ 256 B	—	—

[a] a maximum of $+xB$ for padding need to be added
[b] packet size is reduced by the size of the Integrity Proposal

B. Implementation

Cryptography and highly restricted devices tend to exclude one another at the first glance. Comparing the cipher requirements in Table II with the constraints of the used devices in Table III gives a first idea on this contradiction. As resources – and especially their amount of main memory – are very limited, some thoughts on limitations and cryptographic choices introduce this section. The second part summarizes potential pitfalls and caveats found during the implementation of a minimal G-IKEv2 client.

Limitations and cryptographic choices

In order to maximize compatibility between different implementations, IKEv2 as well as G-IKEv2 support numerous cryptographic functions, which are negotiated during the initialization phase. In order to minimize payloads, exactly one offer will be defined for a minimal implementation. Additionally, the cryptographic function must be chosen carefully in order to honor the characteristics of constrained environments and devices.

Table II shows the impact of popular cryptographic functions on the size of the IKE_SA_INIT and GSA_AUTH messages. It also gives an overview of the size of the cryptographic keys that need to be stored in the SAD to secure the key exchange and the group communication itself. As the client is not mandated to choose the cryptographic function for the group communication, the group communication key server needs to be configured carefully, always having the limitations of the clients in mind.

Any IKEv2 implementation must be able to handle messages of up to 1,280 Byte (see RFC 7815 [4]). This requires a network buffer of at least the same size. Furthermore, the implementation has to be able to store a minimum of two SAs and its corresponding keys – the IKE SA – used to secure the initial key exchange and at least one TEK group SA downloaded from the GCKS. Another necessity for secure communication is a source of randomness in order to generate a Diffie-Hellman key and a nonce for the IKE_SA_INIT request. G-IKEv2 communicates over UDP and thus a network stack implementing ISO/OSI layers 1–4 is required, too. For securing the GSA_AUTH exchange the implementation needs at least one valid cipher suite consisting of one encryption and integrity algorithm, one *pseudo-random function* (PRF) and one Diffie-Hellman group. Given these facts, AES128-CBC for confidentiality, HMAC-SHA256 for integrity, HMAC-SHA1_96 as a PRF and ECP256 for Diffie-Hellman are chosen to be used.

The presented implementation is built on RIOT OS. The main reason for its choice is its lightweight nature with a minimal requirement of 1.5 KB main memory. Additionally, RIOT supports various hardware and includes a network stack fulfilling the requirements as stated above. Finally, various cryptographic libraries targeting embedded systems are available for RIOT. Table III shows the highly constrained devices used for the analysis in section IV.

Pitfalls and caveats

While RIOT solves many problems concerning portability and compatibility of the implementation, its network stack initially did not allow network packets to be encrypted with RIOT's own crypto functions. The problem is the way network data is stored in the packet buffer, which is a

statically allocated section of the main memory and used to save incoming and outgoing network packets. A packet is represented by a linked list of so-called 'pktsnip'-structures, each containing a pointer to the packet's payload and a pointer to the next element in the list. A typical network module (e. g. the UDP module) will prepend a 'pktsnip' to the linked list containing a pointer to its data (e.g. the UDP header). Everything is stored at the first empty segment of packet buffer. While this solution solves the problems that dynamic memory allocation is not recommended and thus should not be used in constrained environments, it makes it impossible to calculate a cryptographic cipher over the packet data because the data is spread over the entire buffer and not linearly aligned. The solution to the problem is to introduce a new function called *gnrc_pktbuf_merge()* which merges the linked list to a single 'pktsnip' and stores the data consecutively into the packet buffer, which can be fed into the cryptographic function.

Another unforeseen challenge is that RIOT's default stack size of 1,024 Byte for Cortex-M CPUs turned out to be too low. This lead to non-deterministic kernel panic in different phases of the key exchange, making the implementation unreliable and distorting measurement results and was solved by increasing the stack size to 2,048 Byte.

IV. Evaluation

The aforementioned implementation is the basis for an evaluation of the minimal G-IKEv2 client. First, the security of the design and the implementation is evaluated before the latter is tested on basis of a typical scenario for the G-IKEv2 protocol – the initial key exchange with a carefully selected set of parameters on various hardware platforms.

A. Security Analysis

As this paper presents a security protocol, analyzing its security is of major interest and significance. The design presented in Section III-A does not weaken the security of its underlying protocols IKEv2 / G-IKEv2. This is accomplished by only removing optional messages and payloads, which keeps the client fully compliant to the RFCs. In order to keep the implementation's security level high as well, we chose only secure cryptographic algorithms which are based on elliptic curves. Additionally, the implementation uses well known cryptographic open source libraries for embedded devices and is available open source[1] allowing security experts to prove the security of the implementation.

B. Minimal system requirements

In order to minimize network and memory overhead, section III-A defined a minimally required subset of messages and content to exchange cryptographic material. However, this still puts some minimal requirements onto the system executing the minimal G-IKEv2 client, which are briefly discussed in the following.

Table III
Available Adruino boards in testbed

Arduino	Architecture	Clock Speed	Flash Memory	SRAM
UNO	ATmega328	16 MHz	32 KB	2 KB
M0 Pro	ARM Cortex-M0+	48 MHz	256 KB	32 KB
Due	ARM Cortex-M3	84 MHz	512 KB	96 KB

A minimal IKE_SA_INIT request has a size of 88 Byte. It consists of the IKEv2 header and several other headers for the different payloads to negotiate the IKE_SA. As shown in Table II, the DH group has the most significant impact on the size of the message.

Another important value is the Nonce for the Diffie-Hellman public value. On the one hand, it is arbitrary, but on the other hand and in order to grant a certain level of security it is recommended to use at least 32 Byte. The use of combined cryptographic modes has a positive impact on the size of the IKE_SA_INIT as it combines the negotiation of encryption and integrity into one proposal.

The GSA_AUTH payload requires a minimum of 98 Byte. As the cryptographic function to ensure encryption and grant integrity have to be same among all group members, they are defined by the GCKS. Thus, any minimal implementation must be able to handle the length of any cipher supported by G-IKEv2, but not necessarily the cipher itself. Of course, the minimally implemented client can refuse to join a group using ciphers that are not supported.

With that, one can deduce a minimal amount of memory needed in order to be able to execute a minimal G-IKEv2 client on a constrained device. At least the largest G-IKEv2 message (usually the GSA_AUTH response[2] as it also contains the GSA and its keys) has to be handled. Additionally, the device must be able to store the SA information plus the cryptographic keys in non-volatile memory to survive reboots or sleeping modes.

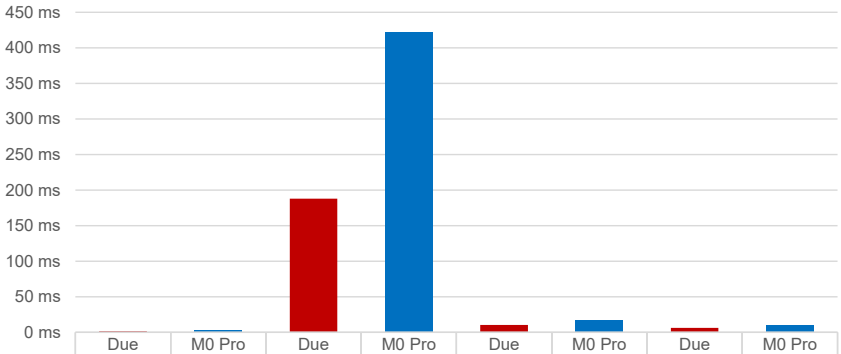
C. Configuration for tests

The testing scenario initiates and carries out a typical G-IKEv2 key exchange in order to join a group managed by a GCKS. The GCKS is implemented using a Cisco CSR1000v server, which comes with a pre-installed G-IKEv2 implementation supporting both server and client mode. It is configured to allow any incoming connection and using pre-shared secrets to grant access to a group.

Three different developer boards are executing the minimal G-IKEv2 client trying to join a communication group handled by the Cisco server. The boards were mainly chosen due to using popular microprocessors and their good

[1] http://www.nm.ifi.lmu.de/projects/embedded/

[2] The GSA_AUTH response originating from a Cisco CSR1000v server has a size of 780 Byte including any header down to the link layer (see section IV-C for further information on the test setup).



	Due	M0 Pro	Due	M0 Pro	Due	M0 Pro	Due	M0 Pro
	Prepare IKE_SA_INIT		Process IKE_SA_INIT		Prepare GSA_AUTH		Process GSA_AUTH	
avg [ms]	1.62	2.62	187.92	421.94	10.29	17.41	6.32	10.53
std. dev. [ms]	0.00	0.00	0.11	0.13	0.00	0.00	0.40	0.00
min [ms]	1.62	2.62	187.72	421.71	10.29	17.41	6.15	10.52
max [ms]	1.62	2.62	188.11	422.18	10.29	17.41	7.26	10.53

Figure 2. Time to prepare and process G-IKEv2 packets on Arduino M0 Pro and Arduino Due

availability. Additionally, all three of them are supported by RIOT. The devices run RIOT, which was compiled with the minimal G-IKEv2 client implementation. Every device is equipped with a WiFi module (namely Espressif ESP8266[3]).

In order to ensure comparability, the G-IKEv2 configuration that can be handled by any of the test devices is specified. While the server supports more than one proposal, the minimally implemented clients propose only a single cipher suite consisting of the following:

- AES128-CBC for confidentiality
- HMAC-SHA1_96 as pseudo-random function
- HMAC-SHA256 for integrity
- ECP256 for Diffie-Hellman

The AES and SHA implementations in use can be found in RIOT's crypto and hashes modules, for elliptic curve Diffie-Hellman (ECDH) the popular micro-ecc[4] library is used. The 32 Byte Nonces sent in the IKE_SA_INIT are generated with tinymt32[5] because the Arduino Due is the only developer board (among the three test devices) equipped with a hardware random generator.

D. Evaluation

To evaluate the performance of the minimal G-IKEv2 implementation, the most interesting values are a) the memory consumption and b) the time necessary to gain access to a group, i. e. to exchange group keys.

Memory Analysis

The implementation itself needs a stack size of 504 Byte. Storing cryptographic material in the SAD (storage for algorithms and keys) and SPD (storage for policies defining in-/outgoing traffic to be secured) requires at least around 140 Byte per group membership and around 210 Byte for securing the connection to the GCKS. Of course, these values depend on the used cryptographic functions and the size of their keys. With a minimal kernel and network implementation, such as the one shipped with RIOT, the need of memory can be stripped down to around 6 KB, which can still be reduced by minimizing the packet buffer.

Table IV
MEMORY REQUIRED FOR THE MINIMAL G-IKEV2 CLIENT

Feature	Required Memory
RIOT kernel (incl. stack)	2,560 Byte
RIOT IPv6 stack	1,024 Byte
RIOT UDP stack	1,024 Byte
RIOT net cache	928 Byte
RIOT packet buffer	1,280 Byte
IKE SA	~ 210 Byte
SAD for 1 group membership	~ 100 Byte
SPD for 1 group membership	40 Byte
Σ	6,142 Byte

This proves the Arduino Uno not being adequate to include both, a network stack and a G-IKEv2 implementation. However, if a full network stack is not required (e.g. for wireless communication on lower layers only), even a device as constrained as the Arduino Uno could theoretically be able to gain access to group keys using a minimal G-IKEv2. A practical proof is yet pending and left for further study.

Performance Analysis

Measuring the time needed to gain access to a group is broken down into four sub-measurements: 1) prepare the IKE_SA_INIT packet and write it into the network stack's buffer, 2) process the server response (this includes calculating the Diffie-Hellman shared secret), 3) prepare the GSA_AUTH request in order to join the group and write it into the network stack's buffer and 4) process the server response. Figure 2 illustrates the results of 20 tests

[3]https://espressif.com/en/products/hardware/esp8266ex/overview
[4]https://github.com/kmackay/micro-ecc
[5]http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/

per device, which are considered to be appropriate as the standard deviation is notably low.

Processing the IKE_SA_INIT response turned out to be the most resource consuming part of the key exchange. The root cause is that this step includes the generation of the Diffie-Hellman shared secret, which is the only asymmetric cryptographic function used in the test case. It consumes more than 99 % of the time of this measurement. The fact that asymmetric functions are performing worse than symmetric ones is well known [15] and thus not surprising. Due to its higher clock speed, the Arduino Due is able to calculate the Diffie-Hellman in just about 44.5 % of the time the Arduino M0 Pro needs. A comparison of the results in [15] and the measurements in this paper reveals that the calculation of elliptic curve Diffie-Hellman values is 3-times (Arduino M0 Pro) or even 7-times (Arduino Due) faster than using elliptic curve based signatures.

V. Summary and Future Work

Secure group management faces a series of challenges in constrained environment, which are not or only insufficiently addressed by current solutions. After specifying the requirements for secure group management, an in-depth analysis of existing solutions reveals a number of open issues or gaps in today's implementations. In order to address these gaps, the solution closest to the requirements of secure group communication is chosen and extended. Concretely, a minimal client for G-IKEv2 has been implemented with extended functionality for group management, opening it for the mass of available IoT solutions. The evaluation of this implementation confirms that a useful performance can be achieved for gaining access to groups and performing re-key operations, while at the same time requiring only a minimum memory consumption. This supports our request for a useful solution in IoT and sensor networks.

Future Work

However, the status of the project, while promising, is only an intermediary state. Ultimately, we want to design a secure group management solution on top of G-IKEv2, which offers identity management within a lightweight AAI, such that authentication and authorization are possible similar as on modern full-fledged devices. This would allow even more functionality at higher security within IoT and sensor network applications.

References

[1] R. Silva, J. S. Silva, M. Simek, and F. Boavida, "Why should multicast be used in WSNs," in *IEEE International Symposium on Wireless Communication Systems 2008*, G. Qu, Ed. IEEE, 2008, pp. 598–602.

[2] T. Guggemos, N. gentschen Felde, and D. Kranzlmüller, "Secure Group Communication in Constrained Networks – A Gap Analysis," in *The 1st 2017 GLOBAL IoT SUMMIT (GIoTS'17)*, Jun. 2017.

[3] B. Weis, V. Smyslov, and Y. Nir, "Group Key Management using IKEv2," Internet Engineering Task Force, Internet-Draft draft-yeung-g-ikev2-11, Mar. 2017, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-yeung-g-ikev2-11

[4] T. Kivinen, "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation," RFC 7815, Mar. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7815.txt

[5] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2013, pp. 79–80.

[6] T. Hardjono and B. Weis, "The Multicast Group Security Architecture," RFC 3740, Mar. 2004. [Online]. Available: https://rfc-editor.org/rfc/rfc3740.txt

[7] S. Camtepe and B. Yener, "Key Distribution Mechanisms for Wireless Sensor Networks: a Survey," Tech. Rep., 2005. [Online]. Available: http://www.cs.rpi.edu/research/pdf/05-07.pdf

[8] L. Salgarelli, M. Buddhikot, J. Garay, S. Patel, and S. Miller, "Efficient authentication and key distribution in wireless IP networks," *IEEE Wireless Communications*, vol. 10, no. 6, pp. 52–61, 2003.

[9] B. Aboba, D. Simon, and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework," RFC 5247, Aug. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5247.txt

[10] T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5246.txt

[11] S. Rafaeli and D. Hutchison, "A Survey of Key Management for Secure Group Communication," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 309–329, Sep. 2003. [Online]. Available: http://doi.acm.org/10.1145/937503.937506

[12] T. Hardjono, S. Rowles, and B. Weis, "The Group Domain of Interpretation," RFC 6407, Oct. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6407.txt

[13] W. Werner, A. Somaraju, S. Kumar, and H. Tschofenig, "Security for Low-Latency Group Communication," Internet Engineering Task Force, Internet-Draft draft-somaraju-ace-multicast-02, Oct. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-somaraju-ace-multicast-02

[14] T. Fossati and H. Tschofenig, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things," RFC 7925, Jul. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7925.txt

[15] M. Sethi, J. Arkko, A. Kernen, and H.-M. Back, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks," Internet Engineering Task Force, Internet-Draft draft-ietf-lwig-crypto-sensors-02, Feb. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-lwig-crypto-sensors-02