No Cookies, just CAKE: CRT based Key Hierarchy for Efficient Key Management in Dynamic Groups

Tobias Guggemos*, Klement Streit†, Marcus Knüpfer†, Nils gentschen Felde*, Peter Hillmann†
*Ludwig-Maximilians-Universität München, †Universität der Bundeswehr München
Munich Network Management Team, Munich, GERMANY
Email: *{guggemos, felde}@nm.ifi.lmu.de †{klement.streit, marcus.knuepfer, peter.hillmann}@unibw.de

***Abstract*—With rapid growth of mobile network linkage, the information exchange between portable and lightweight systems increases heavily. Exchanging confidential information within groups via unsecured communication channels is a high security threat. Consequently, the group participants need a common group key to enable encrypted broadcast messages. Efficient key management of secured group communication is a challenging task, if participants rely on low performance hardware and small bandwidth. Especially, dynamically changing group compositions generate large management expenditure. Considering these requirements, a Group-Key-Management concept including a communication protocol is proposed in this paper. It combines key formation and key distribution functionalists of existing concepts in order to reduce key computation and control message overhead. The lightweight G-IKEv2 protocol in combination with the key exchange concept of CAKE leads to an efficiently integrated solution.**

Keywords*—*GKMP, IKEv2, wireless network, CAKE

I. Introduction

In todays interconnected world, the network linkage is growing rapidly. More and more devices are connected, especially in mobile and resource-constrained networks. In order to cope with the constraints (e. g. limitations on the network link) in such environments, multicasting and group communication is becoming more and more important [1]. Efficiency is achieved by transmitting information only once, but simultaneously to all group members.

Unfortunately, keeping a suitable level of security is challenging in these scenarios. The challenges arise mostly from two aspects of group communication. Firstly, a set of communication peers (referred to as the communication group) must be managed securely. Secondly, the communication among the group members within the group must be secured.

Usually, security in such an environment is facilitated using so-called *Group Keys*. Group keys can be used for encrypted communication within a communication group, but also to preserve other security attributes such as integrity or authenticity. Typically, all *Group Members* possess a symmetric key used to en-/decrypt data. In this paper, encryption of group communication is considered exemplary, without loss of generality.

This work aims at designing a highly efficient, but secure group key management scheme to facilitate secure group communication. The motivation is mainly found by the manifold use of constrained devices in a wide range of applications such as civil, industrial or military use cases. A prominent example for a civil application is car-to-car communication, mainly revealing highly dynamic group formations and wireless network limitations. In contrast, in home automation scenarios (think of smoke detectors on battery) or military applications such as head-mounted units limitations in terms of availability of power, main memory and storage, CPU and network datarates prevail.

Encrypted communication among a set of more than two group members is common to all scenarios. Thus, it seems desirable to share one cryptographic key among the group members in order to encrypt message transfers. Unfortunately, the management of such a key becomes costly quickly due to the dynamics within a group (i. e. members joining or leaving the group rather frequently). Changing the cryptographic material upon every single group management action seems unavoidable, which motivates working towards other than naive approaches. Among general security and usability requirements [2], a special focus during the conceptual phase in this paper is put on:

Forward Secrecy: Whenever a group member leaves the group or is expelled, the member in question must not be able to have access to a valid group key.

Backward Secrecy: Whenever a new group member joins a group, the member in question must not be able to have access to a formerly valid group key before joining.

Key Independence: Having access to one key must not yield the possibility to deduce another member's key.

Collision Free: Additionally, specific to group communication, there must not be a subset of group members that can deduce another member's key(s) by combining their knowledge.

Minimal Trust: A certain trust relationship will be found mandatory, but it shall be subject to minimize.

Low Datarates: The amount of transmitted data shall be minimal in order to facilitate network limited applications.

No 1-to-n Effect: Limited impact of a single membership change on all the other group members is mandatory, meaning not suffering from the 1-affects-n phenomenon, if a single membership change in the group affects all the other group members.

Minimal Delay: The delay imposed by the use of both management actions and cryptographic operations must be minimal.

Minimal amount of key changes and exchanges: The amount of management actions such as exchanging (new) keys shall be limited to a necessary minimum (not implying anything about the total amount of keys in general).

Low calculation complexity: Especially in scenarios exposing CPU limitations, a low complexity of cryptographic calculations is vital, while keeping up the maximally possible security level at the same time.

Compatibility: Clients not capable of supporting CAKE should not be excluded from the communication. Thus, a potential fallback to standardized mechanisms should be supported.

In this paper, a concept is introduced to improve the efficiency and security of group communication in constrained environments. *Group-IKEv2* (G-IKEv2) [3], which is based on *Internet Key Exchange v2* (IKEv2) RFC 7296 [4], is used as communication protocol on the one hand and the group key management concept *Central Authorized Key Extension* [5] on the other hand. Section II introduces related work on secure group communication and evaluates security and efficiency regarding the given scenario with a special focus on the *Logical Key Hierarchy* (LKH). On basis of this, Section III describes the concept of this work combining the lightweight G-IKEv2 and *Central Authorized Key Extension*. Finally, Section IV evaluates the findings and compares with the well-established LKH, before Section V summarizes and concludes this paper.

II. Related Work

Rafaeli et al. [6] 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 *Cryptographic Key Schemes (CKS)* than networking protocols, but some of them are included in *Group Key Management Protocols*. The paper at hand offers the integration of an optimized *Cryptographic Key Schemes* into a centralized management protocol. Thus, the following section is divided in *Group Key Management Protocols* for communication and *Cryptographic Key Schemes* to manage the group key. To our knowledge, none of the approaches provides an efficient and integrated solution, especially with focus on low resource requirements. This is one of the reasons why the *Internet Engineering Task Force* (IETF) started a standardization process for group key distribution in February 2018 [7].

A. Group Key Management Protocols

A high-level definition of *Group Key Management Protocols* (GKMP) and their corresponding architecture is given by the IETF standard body in RFC 2093 [8] and RFC 2094 [9]. The *Internet Security Association and Key Management Protocol* (ISAKMP, RFC 2408 [10]), and *Group Domain of Interpretation* (GDOI, RFC 6407 [11]) have been the first instantiations. The requirements and design of these protocols were derived from multicast architectures of network vendors. Both, peer-to-peer key exchange and Group Key Management were revised for the sake of stronger security properties and better performance, resulting in *Internet Key Exchange v2* (IKEv2, RFC 7296 [4]) and the currently proposed G-IKEv2 [3] for groups.

B. Cryptographic Key Schemes

Centralized CKS comprise a central control authority to manage the group key and to coordinate the cryptographic procedures, often based on a GKMP. In contrast, decentralized techniques share the management of the keys between several instances [12], [13]. Thereby, the generation and distribution of group keys is realized by cooperative instances, which are typical hierarchically ordered. In addition, distributed key agreement procedures delegate the key generation process to not only an individual group member, but to a group of members.

One example is the *Group-Diffie-Hellman* Key Exchange [14], but others exist [6]. All members of a group are organized in a virtual topology, typically into a ring, hierarchies on basis of trees, or just unstructured. In all these schemes, every member of a group shares a common *Transport-Encryption-Key* (TEK).

Another approach is dividing groups into subgroup with individual TEKs. A master within every subgroup takes care of the communication and keys, which allows avoiding 1-to-n effects while re-keying [12]. The downside is requiring repetitive conversions of encrypted messages between the subgroups.

Despite their structured nature, centralized CKS can further be categorized into one of the three subcategories:

- Pairwise keys: Transmission of the group key by the central instance via individual subscriber communication
- Broadcast secret: Transmission of the group key via broadcast instead of individual secured connections
- Hierarchical structure: Coordination of participants in a tree structure with corresponding cryptographic subkeys

The first and most widely recognized CKS ever is defined in the GKMP, which belongs to the category of the pairwise keys. The central server shares an individual secret key with each group member, which is called the *Key-Encryption-Key* (KEK). Subsequently, the server sends the group key to each participant individually encrypted using the KEK. Upon change of the group constellation, the entire group is re-created, leading to high management and communication overheads.

An example for the broadcast secret is the *Secure Lock* (SL) [15], [16] that enables the creation of a group or a re-keying action using a single broadcast message. The SL scheme is based on the *Chinese Remainder Theorem* (CRT) [17], [18], which uses the properties of congruence to encrypt. However, the reduction of communication overhead is obtained by more complex calculations compared to GKMP so that this approach only renders feasible in special scenarios.

A compromise are schemes building on hierarchical structure. A well-known approach is *Logical-Key-Hierarchy* (LKH) [19], [20], which is integrated into GDOI and G-IKEv2. The KEK's and the group participants are maintained in a binary tree. Each node in the tree represents a KEK that is known to the underlying nodes. Maintaining the associated keys of the tree structure increases the management effort, especially the calculation and distribution of internal keys. This approach offers a moderate advantage only in case of repetitive leavings of group members.

Focusing on the motivation for this paper, a centralized scheme with common TEK renders mandatory, especially in order to control and authorize individual members of a group.

In this paper, a combination of the advantages of GKMP, SL and LKH as CAKE [5] with an integration into G-IKEv2 is proposed, allowing for efficient key management.

III. Concept

Targeting highly efficient and encrypted group communication, this paper proposes the combination of lightweight *G-IKEv2* ([1], [3]) for the key exchange and *Central Authorized Key Extension* (CAKE) [5] for the group key management. CAKE's key management is centrally organized and requires a trustworthy *Group Controller* (GC). The GC is responsible for the generation, administration and distribution of the keys and thus requires more computational power than any other (lightweight) group member.

The remainder of this section is organized into subsections inspired by group management operations and patterns:

(A) Client-Server communication based on G-IKEv2
(B) Member Registration on the GC
(C) Group and Group Key Creation
(D) Re-Key of the group
(E) Join of member(s) to a secured group
(F) Leave / Exclude of member(s) from a secured group
(G) Tree Management and Key Addressing
(H) Merging and splitting groups

A. Client-Server communication based on G-IKEv2

G-IKEv2 [3] is used to secure the transmission of cryptographic material for CAKE as it has already proven suitable for constrained devices [1]. G-IKEv2 already supports the establishment of a confidential and authenticated 1-to-1 channel between a client and the GC. It also offers the distribution of *Group Transmission Encryption Key*s (GTEK) and *Group Key Encryption Key*s (GKEK) and thus only requires additional support for CAKE. To communicate securely in a group, every group member has to possess a GTEK used for the communication in the group and a GKEK used to distribute the GTEKs securely. Figure 1 gives an overview about G-IKEv2:

1) **Key Exchange**: A G-IKEv2 key exchange can be divided into two phases:
   a) Establishing an *Initial Security Association* (IKE_SA_INIT): The first two messages from the client to the GC and back establish a *Security Association* (SA) and thus a secure channel between the client and the server (Phase ①: Initialization).
   b) Exchanging keys (GSA_AUTH): Given the secured communication path, the client identifies and authenticates itself and in turn receives transport and key encryption keys (GTEK and GKEK) from the server. The *Group Security Association (GSA) Policy* includes the security parameters (algorithms, lifetime, etc.), while the actual keys are transported within the *Key Download (KD) Payload* (Phase ②: Group Lifetime).
2) **Re-Keying** (GSA_REKEY): Whenever a GTEK or GKEK loses validity (e. g. being outdated), a re-keying action is triggered by the server (GSA_REKEY), which is close to equal to the GSA_AUTH phase (Phase ③: Group Key Refresh).

B. Member Registration on the GC

Each participant P_i registers with CAKE by negotiating an individual key pair (Key_i) with the GC during an IKE_SA_INIT exchange (①). The initial exchange is done with a Diffie-Hellman key exchange, which by design lacks authenticity. A second message GSA_AUTH (②) is used to authenticate both, the client and the GC. Note, that the GSA_AUTH can be used to directly join a group as part of the registration process (see Section III-E).

C. Group and Group Key Creation

On request, the GC randomly generates a GTEK and GKEK. According to G-IKEv2, the GC manages cryptographic material and algorithms for every group. They are stored in the *TEK_SA* and *KEK_SA* databases (see Figure 1).

The GC may decide to create a new group with the new group key and members already registered and authenticated by building a GSA_REKEY payload as follows:

1) The GC constructs a CRT congruency in analogy to the SL scheme, so-called *Lock MX*. Therefore, it uses the individual m_i and Key_i from all participants of the specified group to calculate the Lock MX to encrypt the GKEK (see CRT calculation [17], [18]).
2) The GTEK is encrypted with the GKEK. For the sake of efficiency and security [21], XOR-operations are used for bitwise encryption of the new key tuple with a hashed GKEK. However, any encryption method specified by G-IKEv2 is supported.
3) The keys are embedded into a `CAKE_PRIME` GSA Policy (including the new `KEK_MANAGEMENT_ALGORITHM` called `CAKE`) and a `CAKE_PRIME` KD payload. They are distributed using a single GSA_REKEY broadcast message.

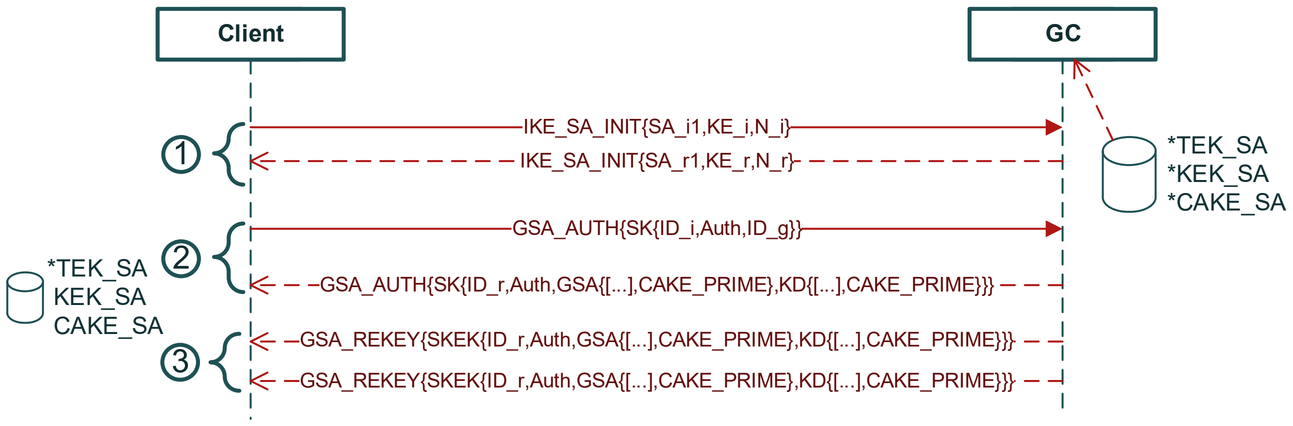
A participant P_i can only "open" the Lock MX, if she possesses a value m_i that was included during the creation of the lock. In consequence, only intended recipients (i. e. group members) care able to read the GKEK and GTEK by solving the CRT.

D. Re-Key of the group

In case the $GTEK$ needs to be renewed, a re-keying action is carried out. The GC generates the keys $GKEK_{new}$ and $GTEK_{new}$, which will be encrypted using the $GKEK_{current}$, embedded into the KD and broadcasted with a GSA_REKEY message. In order to grant forward and backward secrecy, a re-keying action is also carried out every time a member joins or leaves the group.

E. Join of new member(s) to a secured group

If a new participant P_{i+1} wishes to join the group, she sends a GSA_AUTH request including the group ID Id_g she wishes to join. The GC authenticates P_{i+1} and generates an inhomogeneous prime number m_{i+1} for a CRT congruency for P_{i+1}. Additionally, a new GSA policy and KD payload called `CAKE_PRIME` is added, holding m_{i+1}. The use of CAKE is communicated with a new `KEK_MANAGEMENT_ALGORITHM` called `CAKE` within the GSA Policy (see Section 4.5.1.1 in [3]).


Client
GC
*TEK_SA
*KEK_SA
*CAKE_SA
1
IKE_SA_INIT{SA_i1,KE_i,N_i}
IKE_SA_INIT{SA_r1,KE_r,N_r}
2
GSA_AUTH{SK{ID_i,Auth,ID_g}}
GSA_AUTH{SK{ID_r,Auth,GSA{[...],CAKE_PRIME},KD{[...],CAKE_PRIME}}}
*TEK_SA
KEK_SA
CAKE_SA
3
GSA_REKEY{SKEK{ID_r,Auth,GSA{[...],CAKE_PRIME},KD{[...],CAKE_PRIME}}}
GSA_REKEY{SKEK{ID_r,Auth,GSA{[...],CAKE_PRIME},KD{[...],CAKE_PRIME}}}

SA_i1, SA_r1 =	Offered and chosen algorithm and DH Group	Auth =	Authentication:PSK + IDs or Digital Certificate
KE_i, KE_r =	Diffie-Hellman Values	KD =	Key Download
N_i, N_r =	Nonces	GSA =	Group Security Association
SK{...} =	Payload is encrypted and integrity protected	SKEK{...} =	Payload is encrypted and integrity protected with KEK
ID_i, ID_r, ID_g =	Identity (Initiator, Responder, Group)		

Figure 1: G-IKEv2 exchange with CAKE features

The GC also generates $GKEK_{new}$ and $GTEK_{new}$ and embeds the information and keys into an GSA_AUTH sent to the new group member via unicast.

Additionally, a re-key is triggered for any of the former group members. The re-key includes the KD $(GKEK_{new}, GTEK_{new})$ encrypted with the $GKEK_{Current}$. Switching from $GTEK_{old}$ to $GTEK_{new}$ enables the enlarged group (former group plus joined members) to communicate securely.

A mass entry of more than one new participant is equivalent to the process as described before, whereas the $GTEK_{new}$ is send to every new member individually. Alternatively, the new participants can be combined together via a CRT to transmit the $GKEK_{new}$ so that only one message is necessary instead of multiple individual ones. Unfortunately, the latter is only possible if the joining clients are already authenticated. In both cases, two GSA_REKEY messages are broadcasted, one holding the Lock MX for the new clients and one with $(GKEK_{new}, GTEK_{new})$ encrypted with the $GKEK_{Current}$ for the former group members. Thus, an arbitrary number of new members joining a group requires a constant number of messages and thus scales efficiently with the amount of new members.

F. Leave/Exclude of a member from a secured group

Withdrawal of a member from a group can be initiated by the participant herself or be determined by the GC as exclusion. In any case, the presently known $GTEK_{Current}$ and $GKEK_{Current}$ cannot be used, as the expelled participant is in possession of them. To reduce the effort, CAKE uses a reduced CRT system and a ternary tree structure, which is managed by the GC.

Figure 2 illustrates CAKE's tree structure with level A (the root) representing the GKEK and GTEK. Every node represents a pair of keys (m_t and key_t) known by the underlying participants. The actual group members with their personal secrets m_i and key_i are mapped to the leaf nodes of the tree. The designation mX of a node defines a specific m_i for the CRT system.

All pair of keys on the path from the root to the participant must be known by the participant. The tree structure enables efficiency, but its creation can be deferred and only be initialized if necessary. This allows the tree being set up and distributed during a period of low network load. Considering the state of the art, nearly any tree-based scheme ignores this issue and excludes the costs for the tree setup in the evaluation.

Due to their flat structure, trees with more than two subnodes are better suited for larger groups than binary trees. In most scenarios (rarely more than 60 participants and hard to imagine more than 300 [22]), the ternary tree structure is ideal with regard to the size of the tree.

G. Tree Management and Key Addressing

In order to take full advantage of the tree structure, an efficient addressing scheme of nodes or leafs in the tree allows reducing network load. The ID of every key pair is currently an $8x2$ bits address, resulting in a maximum tree depth of 8 and thus 2,187 group members in the group with 3,280 key pairs in the tree. The root key pair has the Id `00`, every parent has the children `01`, `10` and `11` and unused bits are padded with `00`. This allows unique identification of the position of every key within the tree. Additionally, keys required for re-keying actions can be derived easily. Nodes having key pairs not yet included in the tree receive an ID *not* starting with `00`, but e. g. with `11`. After being authenticated with the GC, every participant has his own secret m_i and key_i (see Section III-E), distributed with an address within or outside the tree. This can be implemented by the GC, which may decide to construct and re-balance the tree only if necessary.

Due to the addressing scheme, the protocol for the distribution of keys and performing re-keying requires the implementation of the following actions: I.) Downloading key pairs on the path to the root from the GC, II.) Re-Addressing of

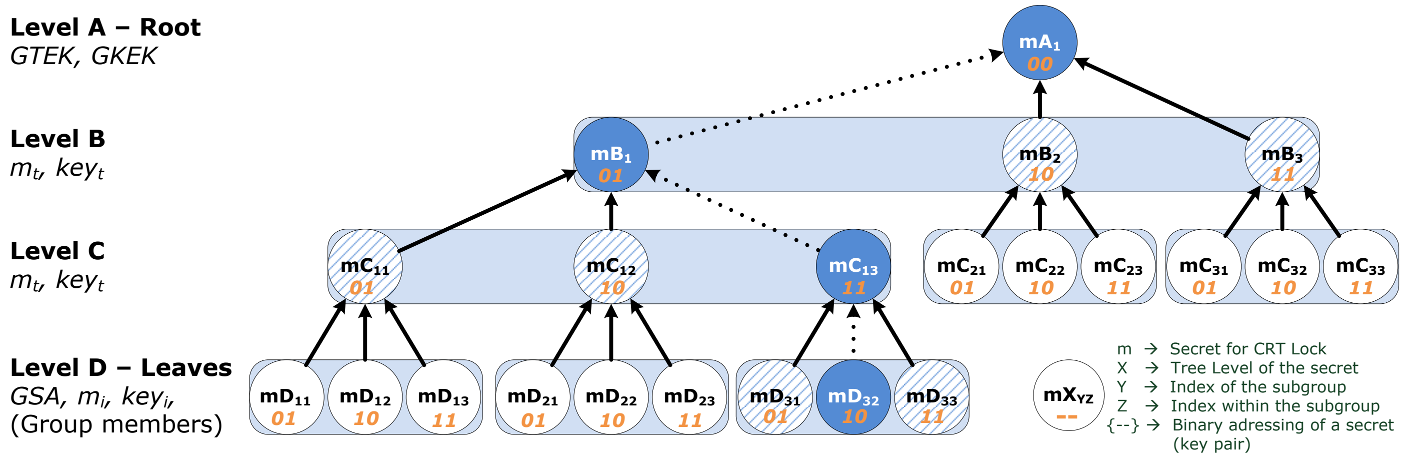


Figure 2: Ternary tree structure to manage the keys and to reduce the calculation effort by withdrawal.

```
 0        1        2        3
+--------+--------+--------+--------+
|   CAKE Key Id   |     RESERVED    |
+--------+--------+--------+--------+
| Size CRT key_id |    Size m_id    |
+--------+--------+--------+--------+
~             CRT (key_id)          ~
+--------+--------+--------+--------+
~      m_id (encr with key_id)      ~
+--------+--------+--------+--------+
```

(a) CAKE Keys Substructure

```
+--------+--------+--------+--------+
|      # Keys     |     # Leaves    |
+--------+--------+--------+--------+
~    CAKE_Leaves_Substructure []    ~
+--------+--------+--------+--------+
~     CAKE_Keys_Substructure []     ~
+--------+--------+--------+--------+
```

(b) CAKE Leave Array

Figure 3: Exemplary CAKE Structures for G-IKEv2

Keys, III.) Receiving updates of key pairs and IV.) Re-Keying upon removal of group members.

Similar to the payloads already defined for *LKH Download Type* (see Section 4.8.3 in [3]), four CAKE Download Types and three substructures holding the array elements are defined. As shown above, any message can be distributed securely as broadcast and thus all download types are designed as arrays. The elements of the arrays are indexed with the identity of the key pair in the tree. Using the unique tree IDs, the nodes can detect if they are affected by the action or not by comparing address prefixes.

CAKE_DOWNLOAD_ARRAY can be embedded in GSA_AUTH or GSA_REKEY message and holds a list of keys, transported within a *CAKE Keys Substructure* (see Figure 3a). A CRT for m_i and key_i is built using the key pairs of the children. The substructure holds the address of the key pair, the lengths of the resulting CRTs and the CRTs themselves. The tree has to be distributed bottom up. A client is able to solve the Lock MX if he is in possession of one children key pair.

CAKE_UPDATE_ARRAY has the same content as CAKE_DOWNLOAD_ARRAY, but defines that the distributed keys were distributed before and, thus, need to be updated in the client's CAKE_SA.

CAKE_READRESS_ARRAY gives a key pair a new address in the tree and can be used to include nodes to the tree or to re-arrange subtrees (e.g. when the tree is re-balanced). The array holds a list of *CAKE Readdress Substructures* holding a tuple (Id_{old}, Id_{new}). If a node receives re-addressing of one of the keys on his path to the root, it needs to update all keys on its path down the tree.

CAKE_LEAVE_ARRAY - When excluding a node, a new root key pair is created and used to encrypt the GKEK which is embedded in a KD Payload (see Section III-D). The leaving node is in possession of all key pairs on the path to the root (see Figure 2: node mD32, dark, binary address: 00-01-11-10), all of which need to be excluded from the Lock MX generation. Instead, the mX located next to a marked node on the same level are used (see hatched nodes in Figure 2). The same is done for the path to all key pairs, the leaving client is in possession of. The new key pairs are embedded in a *CAKE Keys Substructure* (see Figure 3a) which in turn is embedded in the *CAKE_LEAVE_ARRAY* (see Figure 3b). Besides the Key Array, it also holds an array of leaving nodes which size is at least one. This mechanism allows excluding multiple nodes with only one message.

Please note that the GC may choose I.) to distribute all new key pairs in one *CAKE_LEAVE_ARRAY*, or II.) to distribute only the root key pair and update the keys on the path later with *CAKE_UPDATE_ARRAY*.

H. Merging and splitting groups

Merging existing groups requires re-keying based on the currently used GKEKs of the groups, announcing a new and common $GTEK_{new}$. One message per group to be merged (2 at a minimum) has to be issued.

Splitting groups is done by re-addressing within the key tree and building two Lock MX including the new keys, one for each group. The number of messages can be as low as one, but is heavily dependent on the previous tree structure.

Both merging and splitting require deletion of the old group keys, which can be done using a *Delete Payload* specified in [3]. Please note that this is not resistant against malicious group members. In the cryptographic community, this problem is referred to as *Post Compromise Security*, which is a yet unsolved problem and will be subject of further studies.

Table I: Comparison of CAKE with LKH and traditional GKMP (represented by G-IKEv2), with special regards on cryptographic overhead. The keys are defined for AES with 16 Byte keys.

	Register/Join	**Mass Join**[2]	**Key Download/Update**[2]	**Tree Operation**	**Leave**[2]
Networking (in Bytes)	KD Payload[1]: 12 Byte				
GKMP	KEK: 16 TEK: 16 Key: 16	p Messages with: KEK: 16 TEK: 16	n Messages with: KEK: 16 TEK: 16	*undefined*	$n-1$ Messages with: KEK: 16 TEK: 16
LKH	KEK: 16 TEK: 16 Key: 16	p Messages with: KEK: 16 TEK: 16	1 Message with[3]: Hdr: $(4+(n-1)*12)$ Keys: $(n-1)*16$	*same as Key Download*	1 Message with[3]: Hdr: $4+log_2(n)*8+\sum_{i=1}^{log_2(n)-1} i*8$ Keys: $\sum_{i=1}^{log_2(n)-1} i*16$
CAKE	KEK: 16 TEK: 16 Prime: 17 Key: 16	1 Message with: Hdr: 16 TEK: 16 KEK[5]: $\lvert CRT(p)\rvert$	1 Message with[3]: Hdr: $4+n*12$ Keys[5]: $(log_3(n)-1)*\lvert CRT(3)\rvert$ Primes: $(log_3(n)-1)*3*17$	1 Message with[3]: Hdr: $(4+8)$ Key[5]: $\lvert CRT(3)\rvert$ Primes: $3*17$	1 Message with: Hdr: $12+log_3(n^2)*8$ TEK: 16 KEK[5] $\lvert CRT(log_3(n^2))\rvert$
Computation					
GKMP[4,7]	GC: $O_K(1)$ Cl: $O_K(1)$	GC: $O_K(p)$ Cl: $O_K(1)$	GC: $O_K(n)$ Cl: $O_K(1)$	*undefined*	GC: $O_K(n-1)$ Cl: $O_K(1)$
LKH[4,7]	*see GKMP*	GC: $O_K(p)$ Cl: $O_K(1)$	GC: $O_K(2^{log_2(n)+1})$ Cl: $O_K(log_2(n)+1)$	*same as Key Download*	GC: $O_K((log_2(n)+log_2(n-1)))$ Cl: $O_K(log_2(n))$
CAKE[6,7]	*see GKMP*	GC: $O_L(p)$ GC: $O_K(1)$ Cl: $O_L(p)$ Cl: $O_K(1)$	GC: $O_L(3*\frac{n-1}{2})$ GC: $O_K(1)$ Cl: $O_L(3*log_3(n))$ Cl: $O_K(1)$	GC: $O_L(3)$ GC: $O_K(1)$ Cl: $O_L(3)$ Cl: $O_K(1)$	GC: $O_L(log_3(n^2))$ GC: $O_K(1)$ Cl: $O_L(log_3(n^2))$ Cl: $O_K(1)$

[1] Required for every Key distributed with G-IKEv2 [2] n being Group Members, p number of members joining or leaving [3] KEK and TEK is carried as in GKMP
[4] GC performs *encrypt* and Client performs *decrypt* [5] $|CRT(i)|$: size of CRT with i elements in Bytes [6] O_L: Complexity of creating/solving Lock MX.
[7] O_K: Complexity of encryption/decryption of keys.

IV. Evaluation

Having a sound concept at hand, this section evaluates CAKE under the following three aspects: Firstly, a (theoretical) comparison of the computational complexity as well as networking load of CAKE, LKH and traditional GKMPs is carried out. Secondly, an implementation of CAKE for RIOT OS proofs both its lightweight nature and its applicability in constrained scenarios. Given the result, the section will close by evaluating CAKE against the requirements as stated in Section I.

A. Comparison with LKH and GKMP

Table I compares the networking and computation overhead of CAKE with a traditional GKMP system (in that case G-IKEv2) and LKH. As long as there are no clients leaving the group, the simple GKMP mechanism performs optimal for both networking and computation. However, GKMP performs badly in terms of number of messages and computations as soon as *Forward Secrecy* is required on client exclusion, which is the major benefit of LKH and CAKE.

Although CAKE requires a pair of keys (m_i and key_i) to be sent when distributing the tree, it can outperform the LKH mechanism introduced in G-IKEv2. The amount of key headers is equal in both systems. Unfortunately, LKH tree entries need to be transported multiple times decreasing its efficiency (for better insights to LKH in G-IKEv2, we recommend Appendix A of [3]). Using a CRT system, CAKE offers the distribution of keys using a single message. Although, the size of the resulting Lock MX increases linearly (see Table II), it still decreases the necessary protocol information heavily.

CAKE also reduces the demand for computational power on the client-side. Instead of carrying out multiple decryption operations (as for example LKH would do), the client has to perform one single modulo and one decrypt operation only. Nonetheless, this comes at the price of storing more cryptographic material (m_i, key_i) compared to LKH where only key_i has to be stored for every node in the tree.

B. Performance on constrained hardware

CAKE is implemented on basis of RIOT OS [23]. RIOT OS is lightweight by nature with a minimal requirement of 1.5 KB of main memory. Additionally, RIOT is open source and supports a wide variety of hardware. Finally, important cryptographic libraries targeting embedded systems are available for RIOT.

In order to grant realistic conditions, the IOT-LAB [24] is used for the evaluation. The GC and any of the clients is executed using an IOT-LAB M3 board, which is equipped with a 72 Mhz CPU and 64 kB SRAM.

The focus of this evaluation is group management and the corresponding key distribution processes. Please note, the initial key exchange using G-IKEv2 is already published in [1].

1) Memory requirements: On basis of the highly constrained resources of the IOT-LAB M3 nodes, a GC and a group comprising 14 clients can be successfully created and managed. The server has to reserve memory for at least all keys in the tree including the nodes, while the clients have to reserve memory for the keys on their path to the root (not more than 7 key pairs, limited by the address space as seen above). The GC requires 2,900 Bytes of data being stored per participant, including keys, IP addresses, memory for CRT calculations and tree operations. Participants require 2,900 Byte per connection to a GC. According to the design principles of RIOT OS memory – including network buffers – needs to be statically reserved.

2) Computational Costs for CRT: Most of the CAKE-actions require a single GSA_REKEY message carrying G-IKEv2 payloads (see [3]). The most expensive operation according to [1] is the IKE_SA_INIT message triggering the

Table II: Required time for Lock MX operations with i elements. For comparison, the time to encrypt and decrypt the key hierarchy of LKH with tree depth i is shown. The number of clients is 3^i for CAKE and 2^i for LKH.

i	Create Lock MX	Solve Lock MX	Size Lock MX	LKH Enc	LKH Dec
1	280,082 μs	88 μs	41 Byte	125 μs	201 μs
2	572,785 μs	189 μs	84 Byte	188 μs	302 μs
3	822,851 μs	275 μs	124 Byte	250 μs	404 μs
4	1,130,065 μs	374 μs	165 Byte	312 μs	505 μs
5	1,377,708 μs	484 μs	206 Byte	374 μs	607 μs
6	1,539,600 μs	604 μs	247 Byte	437 μs	708 μs
7	1,909,062 μs	750 μs	288 Byte	499 μs	809 μs
8	2,231,764 μs	904 μs	328 Byte	562 μs	911 μs
9	2,544,507 μs	1,072 μs	369 Byte	624 μs	1,013 μs
10	2,751,188 μs	1,243 μs	410 Byte	686 μs	1,114 μs
11	3,134,233 μs	1,433 μs	451 Byte	749 μs	1,215 μs
12	3,387,458 μs	1,632 μs	492 Byte	811 μs	1,316 μs
13	3,705,136 μs	1,858 μs	533 Byte	874 μs	1,418 μs
14	3,974,770 μs	2,081 μs	573 Byte	935 μs	1,520 μs

Diffie-Hellman key exchange. Runtime measurements on IOT-LAB M3 nodes showed comparable computation times as the times given for the Arduino Due in [1].

In terms of computational cost, creating and solving the Lock MX on both the server and the client are the most interesting new features of CAKE. This includes the creation of the Lock MX with various tree depths and the time to resolve it on client side. The results are shown in Table II.

Further, *mass joins* (see Section III-E), where the numbers of elements within in the CRT is the number of participants joining simultaneously, yield interesting results, too. It is evident that the computation of the new keys increases with the number of elements in the CRT, which is mainly caused by the larger size of the Lock MX. The measurements show that especially the GM can benefit. It simply calculates one modulo operation, enabling very low computation time, even when the number of elements within the Lock MX is high.

For comparison, the costs for encrypting and decrypting keys within an LKH tree are shown in Table II. It can be seen that even though the AES implementation is highly optimized, solving the Lock MX scales similar to decrypting the keys within the LKH tree. However, lowering network load with CAKE comes at the price of computational overhead for creating the Lock MX, which scales worse than LKH. Optimizing the Lock MX implementations will be part of further studies.

C. Fulfillment of requirements

Besides the practical applicability and its pros and cons in comparison to especially LKH, reviewing the initial design goals and requirements shows completeness. A detailed design explanation and security assessment can be found in [5].

Forward & Backward Secrecy: Re-keying upon every group management action (i. e. a member joining or leaving the group) grants both forward and backward secrecy.

Key Independence & Collision Free: Using the CRT, these requirements are fulfilled by design.

Low Datarates & No 1-to-n Effect: The ternary tree structure reduces the number of keys stored and sent, allowing CAKE to re-key with one single message and thus a reduction to the per-packet overhead.

Minimal amount of key changes and exchanges: The combination of the ternary tree structure and the addressing scheme guarantees a minimal amount of key changes and exchanges.

Minimal Delay & Low calculation complexity: As shown before, CAKE requires a minimum number of messages to be sent upon group changes. Using the CRT for group leave actions, the demand for compute power on the client-side can be reduced to only one modulo operation,

Compatibility: This requirement is met by using the G-IKEv2 protocol. Any client not capable of the CAKE features can still participate in the group on basis of standard re-keying mechanisms, while the optimizations can still be applied to any other group member. Additionally, the use of G-IKEv2 helps meeting common security requirements, as they are well studied and discussed by standardization bodies. CAKE simply optimizes the transportation and calculation of keys, leaving the security parameters of G-IKEv2 untouched.

Minimal Trust: With a variety of authentication mechanisms supported by G-IKEv2, the need for minimal trust can be achieved on a per-scenario-basis.

V. Conclusion

The proposal of a CRT-based key hierarchy for efficient key management in dynamic groups is based on the combination of the lightweight G-IKEv2 protocol in combination with CAKE for the key exchange. The main goal to reduce the network load to a minimum is achieved at the cost of storage space for additional cryptographic key material. Additionally, computationally demanding (cryptographic) operations are delegated to the group controller, relieving the potentially less powerful group members. The CRT-based key hierarchy together with a ternary key tree structure and the well-formed addressing scheme has particularly shown advantageous in the area of secure group communication among highly constrained group members.

As per now, the networking overhead of the LKH extension in G-IKEv2 is highly inefficient. Thus, the optimization of LKH in special scenarios is subject of future work, which will allow for a more comprehensive and technical comparison of LKH and CAKE. Besides further improvements of the CAKE prototype, especially its memory consumption, the evaluation has proven the CAKE-implementation for building and solving the Lock MX worth investigating and subject of future improvements. Lastly, a more detailed analysis of solving the problem of *Post Compromise Security* when merging and splitting groups is of major interest.

Acknowledgment

We thank the MNM-Team for all the helpful comments and discussions while writing this document. We also thank our students Edgar Goetzendorff, B.Sc., and Mehdi Yosofie, B.Sc., for assisting us during the development of the prototype and the helpful discussions.

References

[1] N. gentschen Felde, T. Guggemos, T. Heider, and D. Kranzlmüller, “Secure Group Key Distribution in Constrained Environments with IKEv2,” in *Conference on Dependable and Secure Computing*. Taipei, Taiwan: IEEE, 2017.

[2] E. Omara, B. Beurdouche, E. Rescorla, S. Inguva, A. Kwon, and A. Duric, “Messaging Layer Security Architecture draft-omara-mls-architecture-01,” 2018. [Online]. Available: https://tools.ietf.org/html/draft-omara-mls-architecture-01 (Access Date: 01 May 2018).

[3] B. Weis and V. Smyslov, “Group Key Management using IKEv2,” Internet Engineering Task Force, Internet-Draft draft-yeung-g-ikev2-14, 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-yeung-g-ikev2-14 (Access Date: 01 May 2018).

[4] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, “Internet Key Exchange Protocol Version 2 (IKEv2),” 2014, RFC7296. [Online]. Available: http://tools.ietf.org/rfc/rfc7296.txt (Access Date: 01 May 2018).

[5] P. Hillmann, M. Knüpfer, and G. Dreo Rodosek, “CAKE: Hybrides Gruppen-Schlüssel-Management Verfahren,” in *10. DFN-Forum Kommunikationstechnologien*, P. Müller, B. Neumair, H. Raiser, and G. Dreo Rodosek, Eds. Bonn: Gesellschaft für Informatik e.V., 2017, pp. 31–40. [Online]. Available: https://dl.gi.de/handle/20.500.12116/476 (Access Date: 01 May 2018).

[6] S. Rafaeli and D. Hutchison, “A Survey of Key Management for Secure Group Communication,” *ACM Comput. Surv.*, vol. 35, no. 3, pp. 309–329, 2003. [Online]. Available: http://doi.acm.org/10.1145/937503.937506 (Access Date: 01 May 2018).

[7] N. Sullivan, S. Turner, B. Kaduk, and K. Cohn-Gordon, “Messaging Layer Security,” 2018. [Online]. Available: https://datatracker.ietf.org/wg/mls/about/ (Access Date: 01 May 2018).

[8] H. Harney and C. Muckenhirn, “Group Key Management Protocol (GKMP) Specification,” RFC 2093 (Experimental), Internet Engineering Task Force, Tech. Rep. 2093, 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2093.txt (Access Date: 01 May 2018).

[9] H. Harney and C. Muckenhirn, “Group Key Management Protocol (GKMP) Architecture,” Internet Engineering Task Force, 1997, RFC2094. [Online]. Available: http://tools.ietf.org/rfc/rfc2094.txt (Access Date: 01 May 2018).

[10] D. Maughan, M. Schertler, M. Schneider, and J. Turner, “Internet Security Association and Key Management Protocol (ISAKMP),” 1998, RFC2408. [Online]. Available: http://tools.ietf.org/rfc/rfc2408.txt (Access Date: 01 May 2018).

[11] B. Weis, S. Rowles, and T. Hardjono, “The Group Domain of Interpretation,” 2011, RFC6407. [Online]. Available: http://tools.ietf.org/rfc/rfc6407.txt (Access Date: 01 May 2018).

[12] Y. Challal and H. Seba, “Group key management protocols: A novel taxonomy,” *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 2, no. 10, pp. 3620 – 3633, 2008.

[13] S. Rafaeli, “A decentralised architecture for group key management,” LANCASTER UNIVERSITY, Tech. Rep., 2000.

[14] M. Steiner, G. Tsudik, and M. Waidner, “Key Agreement in Dynamic Peer Groups,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, 2000.

[15] G.-H. Chiou and W.-T. Chen, “Secure broadcasting using the secure lock,” *IEEE Transactions on Software Engineering*, vol. 15, no. 8, pp. 929–934, 1989.

[16] C. J. Antosh and B. E. Mullins, “The scalability of secure lock,” in *IEEE International Performance, Computing, and Communications Conference*, vol. 27, no. 1, 2008, pp. 507–512.

[17] M. Zheng, G. Cui, M. Yang, and J. Li, “Scalable Group Key Management Protocol Based on Key Material Transmitting Tree,” in *Information Security Practice and Experience*. Springer, 2007. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72163-5_23 (Access Date: 01 May 2018).

[18] G. Xu, X. Chen, and X. Du, “Chinese Remainder Theorem based DTN group key management,” in *IEEE International Communication Technology (ICCT)*, vol. 14, no. 1, 2012, pp. 779–783.

[19] N. Sakamoto, “An efficient structure for lkh key tree on secure multicast communications,” in *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, vol. 15, no. 1, 2014, pp. 1–7.

[20] Z. Liu, Y. Lai, X. Ren, and S. Bu, “An Efficient LKH Tree Balancing Algorithm for Group Key Management,” in *Proceedings of the International Conference on Control Engineering and Communication Technology (ICCECT)*, vol. 10, no. 2. IEEE Computer Society, 2012, pp. 1003–1005. [Online]. Available: http://dx.doi.org/10.1109/ICCECT.2012.213 (Access Date: 01 May 2018).

[21] C. Matt and U. Maurer, “The one-time pad revisited,” in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, vol. 11, no. 1, 2013, pp. 2706–2710.

[22] Persistent Systems LLC. (2018) Mobile ad hoc networking solution delivers reliable comms, situational awareness, under rough conditions. https://www.prnewswire.com/news-releases/persistent-systems-successfully-demonstrates-flat-320-radio-mpu5-network-300634923.html/ (Access Date: 24 April 2018).

[23] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, “Riot os: Towards an os for the internet of things,” in *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 79–80.

[24] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, “FIT IoT-LAB: A large scale open experimental IoT testbed,” in *World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 459–464.