Key Set Management in Networked Building Automation Systems using Multiple Key Servers

Wolfgang Granzer, Christian Reinisch, Wolfgang Kastner *
Vienna University of Technology, Institute of Computer Aided Automation
Treitlstraße 1-3, A-1040 Vienna, Austria
{w, cr, k}@auto.tuwien.ac.at

Abstract

With the integration of security critical applications into traditional building automation systems, a comprehensive security concept is mandatory. Most important, transmitted data have to be secured using cryptographic techniques. However, even if the used cryptographic algorithms are perfectly secure, the overall security highly depends on the non-disclosure of the used shared secrets. Therefore, this paper targets the management of these shared secrets and the necessary infrastructure used to manage them. Finally, to eliminate a single point of failure in this infrastructure, a redundancy concept featuring multiple key servers is presented.

1. Introduction

For a long time, security in building automation systems (BAS) has been a side issue at best. Yet, times have changed. The integration of new services formerly provided by separate systems (for e.g., access control and surveillance) promises synergies, but significantly tightens security requirements. A comprehensive approach towards security in building automation networks (BANs) requires all types of communication to be secured. This concerns, on the one hand, process data and, on the other hand, management communication. Therefore, an overview of the communication models used in BAS is given and relevant data security classes are presented at the beginning. The main part of the paper then focuses on retrieval and distribution of shared secrets, which is considered to be an integral step towards secured BANs. This includes the definition of multiple key sets, thus covering all communication types common in building automation. Next, a key server infrastructure accompanied by key set management services (key set binding, unbinding and revocation) and their underlying methods for the corresponding key sets are presented. Finally, issues concerning the elimination of single points of failure are addressed. A multiple key server architecture that provides key set forwarding and replication is introduced.

*This work was funded by FWF (Österreichischer Fonds zur Förderung der Wissenschaftlichen Forschung; Austrian Science Foundation) under the project P19673.

2. Communication in building automation networks

Nowadays, functions of BAS are realized by *distributed control applications*. As it is naturally the case in these distributed systems, there is an inherent need to communicate. On the one hand, the distributed control applications need to exchange process data such as sensor or actuating values (*process data communication*). On the other hand, it is desirable to change control (e.g., configuring device setpoints) and communication parameters (e.g., changing the device address). This form of communication is referred to as *management communication*.

2.1. Communication models

To exchange process as well as management data between the involved communication partners, various communication models can be used. A typical model that is based on a point-to-point relation is the *client/server model*. Here, the requesting entity (client) sends a request to the service provider (server). After having received the request, the server executes the desired action and sends a response or an acknowledgment back to the client.

In a *producer/consumer model*, one or more senders (called producers) provide data by making it publicly available. Other devices decide on their own whether they are interested in the data or not. If they are interested, they receive and deliver the data to the corresponding application (consumers). Otherwise they ignore it. A set of senders and receivers that are interested in the same data is called a *communication group*. Since each group member itself is responsible for joining and leaving the group, there is no central instance that manages the group membership (*loose group membership*).

The *publisher/subscriber model* is similar to the above mentioned producer/consumer model. However, each group member has to explicitly join and leave a communication group by sending a (un-)subscription request to a dedicated group coordinator (*strict group membership*). This group coordinator determines whether the requesting device is allowed to join this particular group or not.

978-1-4244-2350-7/08/$25.00 ©2008 IEEE.

Sending and receiving data within a specific communication group is only possible after a successful group join (subscription). After the device leaves the group, it is no longer able to participate in the communication.

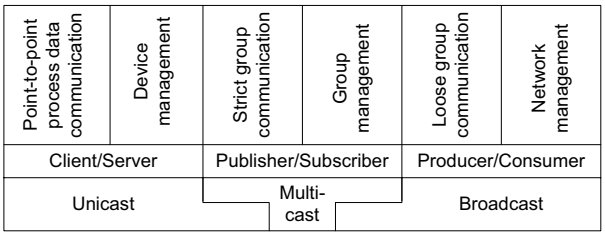

Figure 1. Communication types in BAS

Figure 1 shows the different communication types used in BAS and a mapping of them to the corresponding network protocol mechanisms. Although other mappings may also be valid, the most common one is presented here. In this paper it is assumed that an underlying BAN protocol offers at least a unicast and a broadcast service. An additional multicast service would be especially advantageous for group communication. However, if such a service is not offered by the BAN protocol, either multiple unicasts or a broadcast will be used instead.

All three communication models can be used for process data as well as management data exchange. Therefore, six different *communication types* can be distinguished in the building automation domain:

- *Point-to-point process data communication*: As the name implies, process data is exchanged using the client/server model. For example, a boiler (client) periodically requests the present value of a temperature sensor (server).
- *Loose group communication*: A typical example of this communication type is a lighting system in which multiple light switches are used to control multiple light sources.
- *Strict group communication*: Strict group communication is advantageous in security critical environments where only devices with adequate access rights shall be able to send and receive data of particular interest (e.g., access control system).
- *Device management*: Device management uses the client/server model: A management client starts a so called management session to the particular device which is to be configured or maintained (management server). After the management client has performed the desired management tasks, the session is terminated. Consider, for example, a management device (e.g., an operator work station) intending to change application parameters of a field device.
- *Network management*: Network management refers to management tasks that are addressed to all network members. For example, a network coordinator may want to inform all network members about changed routing information. Typically, network management is based on the producer/consumer model with the management data being distributed to all network members.
- *Group management*: Group management is used to configure and maintain all devices of a particular group at a time. Since only the members of the desired group shall be addressed, group management uses the publisher/subscriber model. A typical example is a group coordinator that informs the group members about a new device joining the group.

2.2. Secure communication

To cater for secure data exchange, a protection of both data, i.e., the exchanged process data (*secure process data communication*) as well as the exchanged management data (*secure management communication*) is mandatory. To protect the exchanged data against malicious interference, it has to be transmitted using a so called *secured channel*. This secured channel protects the communication among multiple entities using physical and/or cryptographic techniques [1]. Depending on the security requirements of the application, a secured channel may guarantee different security objectives. These are data integrity, freshness and/or confidentiality [2]. To achieve the security objectives, *security mechanisms* are necessary. These mechanisms consist of protocols (e.g., authentication) and cryptographic algorithms (e.g., encryption).

For reasons of cost efficiency, the limited resources of embedded devices that are commonly used in the building automation domain are just sufficient for their task. This concerns primarily processing power, (persistent and volatile) memory, power consumption, and network bandwidth. Since security mechanisms are computationally intensive, the realization of security objectives must not exceed the available device resources. The security demands of building automation systems mainly depend on the application type. Therefore, it is not always necessary to guarantee all of above mentioned security objectives. For example, if the non-disclosure of the transmitted data is not a strict requirement, guaranteeing data integrity and freshness may be sufficient. In general, only those security mechanisms that are absolutely necessary to satisfy the security demands of the application shall be implemented (”good enough security”).

According to the security objectives that are guaranteed by a secured channel, the exchanged data can be classified into the following *data security classes*:

- *Plain data*: Plain data is not secured at all. In order to avoid an unauthorized manipulation of plain data, the transmission channel must be secured physically. This also includes non-cryptographic techniques (e.g., tamper resistance). As this work targets mainly key management, these aspects are not discussed further.
- *Protected data*: Protected data means that only data integrity is guaranteed. Additionally, the data origin can be verified.

- *Trusted data*: Protected data that is classified as trusted data is secured in way that data freshness is provided.
- *Confidential data*: Trusted data that is additionally encrypted is called confidential data. It is guaranteed that only authorized entities are able to read the clear text version of the confidential data. The ultimate goal of confidential data is to provide semantic security. Semantic security is a strong level of security where an adversary is not able to gain any information about the clear text, even if being in possession of many encryptions of the same clear text [3].

According to these different data security levels, messages that contain secured data can be classified into *plain*, *protected*, *trusted* and *confidential* messages.

Protected messages can be achieved by adding a signature to the transmitted data. The signature is calculated using *Message Authentication Code (MAC)* algorithms. A MAC algorithm takes a secret key and an arbitrary-length message as input. The output is a fixed length MAC (also called tag). To verify the correctness of a MAC, the receiver recomputes it. If it matches the received MAC, the frame can be considered trusted. Many different MAC algorithms that are suitable for the use in building automation systems exist. Some of the most important ones are *CBC-MAC* [4], *CMAC* [5] and *HMAC* [6].

To generate trusted messages, it must be guaranteed that each message is unique even if multiple messages with the same content are sent. This is done by using a nonce. Such a nonce can be a random value, a strict monotonically increasing counter or a timestamp.

For the creation of confidential messages, an encryption algorithm is necessary. This can be accomplished using either symmetric or asymmetric encryption algorithms. However, the latter ones are in general too resource consuming to be executed on microcontrollers typically used in the building automation domain [7]. For this reason, symmetric encryption algorithms (e.g., AES [8]) have to be used.

Using the above mentioned mechanisms, the assembly of confidential messages requires two separate algorithms, one for signature calculation and one for encryption. Another possibility is the use of *authenticated encryption algorithms*. These algorithms generate the encrypted message as well as the signature simultaneously. Using a secret key, the message and a nonce as input, they output the encrypted message with the corresponding MAC. Two of the most important authenticated encryption algorithms are *Offset Codebook Mode (OCB)* [9] and *Counter with CBC-MAC (CCM)* [10].

In principle, any of the above mentioned mechanisms can be used to set up a secured channel. However, choosing an appropriate mechanism or defining a new algorithm is out of the scope of this paper. Still, the use of *symmetric algorithms* is assumed for the rest of this paper, due to the resource constraints of the embedded devices.

Figure 2 shows the basic principle of a symmetric algorithm. A symmetric algorithm takes two input parameters: the clear text message containing the user data to be transmitted and so called *cryptographic material*. This cryptographic material may consist of one or more secret keys, nonces, initial counter values and timestamps. Taken together this collection is called a *key set*. The output of the algorithm is always the secured message. Depending on the algorithm employed, this secured message is then either a protected, trusted or confidential message. It is important to note that, regardless of the data security class targeted, cryptographic material always has to be shared between all sender(s) and receiver(s).

Today, sophisticated algorithms exist, that provide a good amount of security. However, the overall security of a (building automation) system also highly depends on the non-disclosure of the cryptographic material. In particular, the exchange of key sets over the network is prone to attacks. While several key distribution protocols have been developed for the IT world, they do not meet all requirements of BAS systems. For example, Kerberos [11] is mainly used for the client/server based communication model but no support is given for the other communication types identified for BAS. Therefore, the rest of this paper focuses on *key set management*, which is an essential step towards secure BAS.

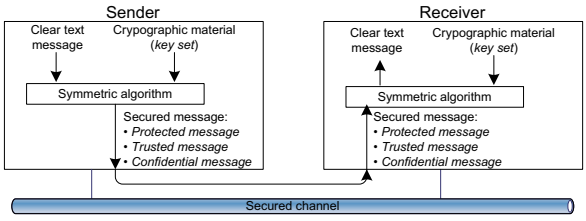

Figure 2. Symmetric algorithm

3. Key sets

Using the same key set for all types of communication is not desirable. This is for two reasons. First, only authenticated entities (i.e., entities that have proven their identities) shall be able to participate in secure communication. Second, using different key sets for different types of communication reduces the number of key set uses. This in turn reduces the chance of attacks that are based on brute force. Therefore, it is desirable to distinguish the key sets into different *key set types* (cf. Section 3.1).

Due to this, a device has to store multiple key sets. In principle, all used key sets could be distributed manually to the device a priori. However, since this solution is obviously inflexible, it is desirable that key sets can be retrieved from a central instance consequently called *key server*. Through the use of a key server infrastructure, it is also possible to change key sets during runtime. In Section 3.2, a possible integration of this key server functionality into BANs is shown.

3.1. Key set types

In order to support secure communication in all kinds of BAS, different *key set types* are distinguished: *Network key sets* protect network messages that are transmitted to all members of the network. *Group key sets* are used to protect communication within a communication group. *Session key sets* are used to secure communication between two devices. Using these three key set types, all six communication types can be secured:

- *Secure network management*: In order to (simultaneously) perform configuration or maintenance of all members of the network in a secure manner, secure communication with all network members is required. This is achieved by protecting the communication service with the network key set. Devices retrieve this network key set during the initialization process, the so called *network join* (cf. Section 4.1).
- *Secure group management*: Group management tasks can be secured using the group key set of the particular group. The group key sets are maintained by the key server that is responsible for the group. All devices are able to retrieve the corresponding group key set during the group join procedure (cf. Section 4.2).
- *Secure device management*: To securely perform device management tasks, the exchanged data is secured using a session key set. This session key set is distributed during the session establishment (cf. Section 4.3).
- *Secure loose group communication*: To securely exchange process data using loose group communication, the process data is secured with the network key set and transmitted to all network members. All network members receive the message. Since each network member possesses the network key set, each member can decide on its own whether being interested in the data or not (cf. Section 4.1).
- *Secure strict group communication*: Securing the process data that is exchanged using strict group communication, the group key set of the particular group is used. Since only group members are in possession of the group key set of the particular group, only they are able to send and receive the process data. As for secure group management, a group key is retrieved during the group join from the maintaining key server. During this subscription process, the key server verifies whether the requesting device's access rights permit the group join (cf. Section 4.2).
- *Secure point-to-point process data communication*: Exchanging process data using the client/server model is performed in sessions. Therefore, this type of communication is secured using a session key set. The distribution and retrieval procedure is similar to the one in case of secure device management (cf. Section 4.3). The most important difference is the lifetime of the used session key set. While a session key set for a device management task is only valid until the management client finishes its task, a session key set for process data communication may be valid for a longer period. This approach is advantageous since process data communication may occur more frequently and over a longer time period.

In order to be able to retrieve any key set in a secure manner, it is a prerequisite that initial key sets called *node key sets* are available on all devices. These node key sets allow the devices to establish a secure connection to the key server and have to be distributed already at installation time. Each device has exactly one node key set that is shared between the device and its key server. However, to decrease the amount of node key set uses (and thus increase security), a *dynamic node key set* that is derived from the node key set is used. Since a new dynamic node key set is generated for each key establishment procedure, a dynamic node key set is only valid during a single key establishment process. However, devices must agree on a common mechanism to generate the dynamic node key set. This can, for example, be a monotonically increasing counter or a (exchanged) nonce, which are then used by both partners to derive the dynamic node key set [2].

3.2. Key server infrastructure

The management of the key sets requires a *key server infrastructure*. Presently, the key server infrastructure of available BAS is mostly rudimentary. In LonWorks [12] as well as in KNX/EIB [13], there is no dedicated key server that supports the alteration of shared secrets (called domain keys in LonWorks and access keys in KNX/EIB) during runtime. Rather, shared secrets can only be uploaded directly to the devices using a configuration tool. In BACnet 2004 [14], a single key server is foreseen for retrieval of so called session keys during runtime [2]. The use of multiple key servers is not defined. In BACnet Addendum g [15], the BACnet 2004 security services were replaced to overcome several security flaws. Amongst others, the use of multiple key servers became possible, however with the implementation and synchronization details still not being specified. In ZigBee [16], the optional key management functionality is provided by a so called trust center. The use of multiple trust centers in a single ZigBee network is not possible.

Clearly, a single key server being responsible for the key management of the whole network introduces a single point of failure. To overcome this limitation, *multiple key servers* are necessary. BANs are divided into different network segments. In this paper, it is assumed that these network segments are arranged in a tree-structure of arbitrary depth. To avoid a single point of failure, each network segment has a dedicated key server that is responsible for processing all incoming key set management requests within its network segment. This decision brings along the advantage that only one network segment is affected in case of an attack or a key server failure.

As shown in Figure 3, the key server functionality can be implemented in two ways. First, it is possible to add

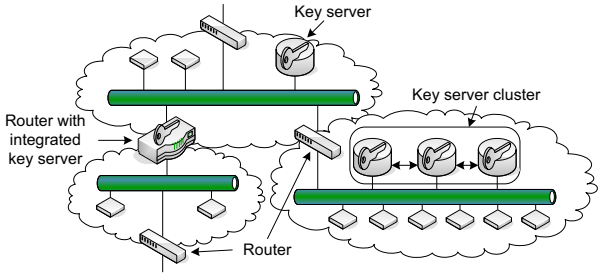

Figure 3. Key server distribution

a dedicated key server to each network segment. Second, an existing device (e.g, a router that interconnects the network segment with its parent segment or a network coordinator) can be extended with the key server functionality.

Although the use of one key server per network segment clearly confines the effects of an attack or failure within this segment, a single point of failure (at least for the particular segment) remains. Therefore, a single key server is only advantageous for small network segments and in case of applications with relaxed requirements regarding robustness and availability. If network segments consist of many devices (e.g., a backbone or wide-ranging field network) and high service availability is mandatory (e.g., alarm systems), this single key server approach is insufficient. To counter this problem, the single key server has to be replaced by a so called *key server cluster*. A key server cluster consists of multiple key servers that are conjointly responsible for key set management within one network segment. Obviously, this key server infrastructure requires some form of synchronization and key exchange among the servers. These details will be discussed in Section 5.

For the remainder of this paper, it is assumed that each network segment disposes of a key server cluster which consists of one or more key servers. Since the use of a key server cluster is completely transparent to the devices (i.e., the key server cluster is perceived as a single key server by the devices), all key set management services are independent from the underlying key server infrastructure. These key set management services are presented in the following section.

4. Key set management services

To be able to manage the key sets in a secure manner, three different services have to be provided by the key server infrastructure. First, a device must be able to set up a secure relation to the desired communication partner(s) (*secure binding*). Consider, for example, a device that wants to communicate securely with another device (i.e., using the client/server paradigm) or with the members of a particular group (i.e., using the producer/consumer or the publisher/subscriber model). At the beginning, the device sends a secure binding request to the key server. The key server verifies the request and sends a response back to the requesting entity. This response contains the corresponding key sets for the requested communication type. These key sets are then used to set up the secured channel between the desired communication partner(s).

Second, a *revocation* service is necessary. This service is used in two different situations. On the one hand, it may be necessary to actively exclude an entity from a communication relation. This is necessary if a device has been compromised or if a device wants to release the communication relation again. On the other hand, the revocation mechanism can also be used to limit the lifetime of a key set by revoking it regularly. During the revocation process, the old key set is declared void and a new one is distributed to the remaining communication partners.

In general, the lifetime of key sets shall correlate with the number of key set uses (cf. Figure 4). Key sets of communication types that are used more often shall be revoked more frequently than the key sets belonging to communication relations that are used rather rarely. However, since revocation as well as generation and distribution of new key sets are no trivial tasks, it is not always possible to revoke key sets as frequently as required for maximum security.

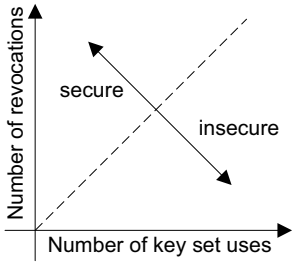

Figure 4. Revocation vs. number of uses

The third key set management service provides a mechanism to release a secure communication relation (*secure unbinding*). Consider, for example, that an entity wants to close a session or leave a communication group. This is done by sending an unbinding request to the key server. From the point of unbinding on, the leaving device must be precluded from all communication within the group. Therefore, the current key set must be revoked and a new one has to be distributed to the remaining communication partners. Thus, a device that wants to resume the communication with the group must perform a secure binding again.

Figure 5 shows the notation that is used for all following diagrams.

4.1. Network key set management

To be able to securely manage the devices within a network segment, three different services are required (cf. Figure 6). It is assumed that a device knows its unique identifier (UID) (e.g., a serial number which is unique within the whole installation) and its node key set. Furthermore, it is assumed a key server cluster consisting of

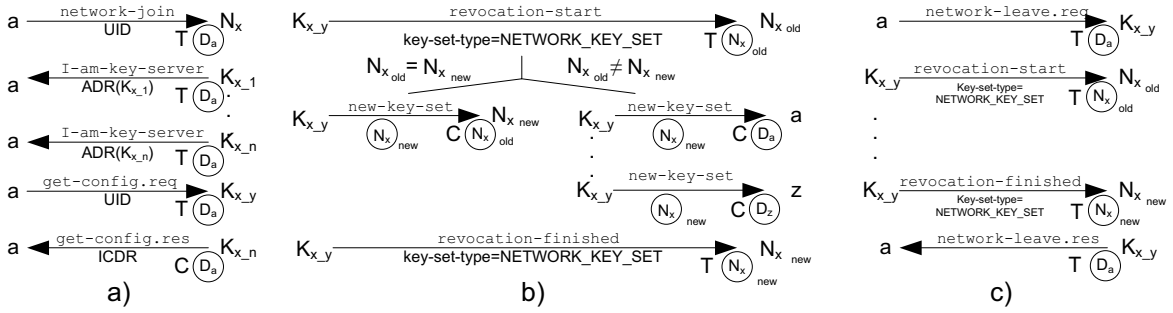

Figure 6. Network join (a), network key set revocation (b) and network leave (c)

a,b,c,...	Field device.
N_x	All members of network segment x.
K_{x_y}	y'th key server device of key server cluster within N_x.
G_x	All members of communication group x.
ADR(X)	Address of a device, group or network segment.
D_a	Dynamic node key set of device a.
N_x	Network key set of network segment N_x.
G_x	Group key set of group G_x.
S_{ab}	Session key set shared between device a and b.
x service content Z KS Y	Device x sends a message to Y. Y can be all members of a group, network or a single device. The message is secured with KS and is an instance of message security class Z (P: protected, T: trusted, C: confidential).

Figure 5. Notation

at least one key server exists in the network segment.

First, a new device must be able to join a network segment (*network join*). As shown in Figure 6, a device a sends a `network-join` message containing its UID to all members of the network segment to start the network join. The message is secured using the dynamic node key set with a security class of at least "trusted". If the message is not secured at all or its security class equals "protected", an adversary is able to replay old `network-join` messages thus causing unwanted network traffic since all key servers send a response. If data freshness is guaranteed (i.e., the message is trusted), these replayed messages can be identified and discarded. Therefore, for the rest of this paper, all key set management messages that do not contain confidential data are trusted messages.

After reception of the request, each key server within the key server cluster responds with a `I-am-key-server` message. This message contains the device address of the key server and is again secured using a dynamic node key set. After having received these responses, the joining device is aware of the addresses of all key servers within its network segment. To retrieve its initial configuration parameters, a sends a `get-config.req` message to one of the available key servers. The key server can be chosen randomly, based on transmission delay measurements or on optional quality-of-service parameters integrated into the `I-am-key-server` message (e.g., load balancing). Again, this message contains the UID and is secured using a dynamic node key set. The key server responds with a `get-config.res` message. It is secured using a dynamic node key set and contains a so called *Initial Configuration Data Record (ICDR)*. The ICDR contains different configuration parameters such as the device address, the network address, and most important the network key set of the corresponding network segment. To avoid an unwanted disclosure of the transmitted key set, this message has to be confidential. For the rest of this paper, all network management messages that are used to transmit key sets are confidential messages.

As from now, the new device is able to communicate within the network segment using the values of the received ICDR. Furthermore, the device is able to process incoming secured network management messages using the network key set.

To start the revocation process of a network key set, the key server intending to revoke the network key set sends a `revocation-start` message including a parameter `key-set-type=NETWORK_KEY_SET` to all network members. This message is secured with the current (old) network key set and indicates that a revocation process is under way. From now on, sending messages secured with the old network key set and distribution of the old network key set is prohibited. After having sent the message, the key server generates a new network key set and distributes it to all network members.

However, the approach of securing the new network key set with the old one is only applicable as long as the set of network members before and after the revocation process is identical. If, for example, the revocation is started due to the detection of a compromised device or due to a network leave, the old network key set can no longer be used to securely distribute the new one. One approach to distribute the new network key set would be to send separate messages, each secured with the dynamic node key set, to all remaining group members. In networks with a high node count, this can clearly lead to considerable network traffic. Another simple solution would be to send just a single message which contains the new network key multiple times. In particular, the new network key is secured with the dynamic node key set of each

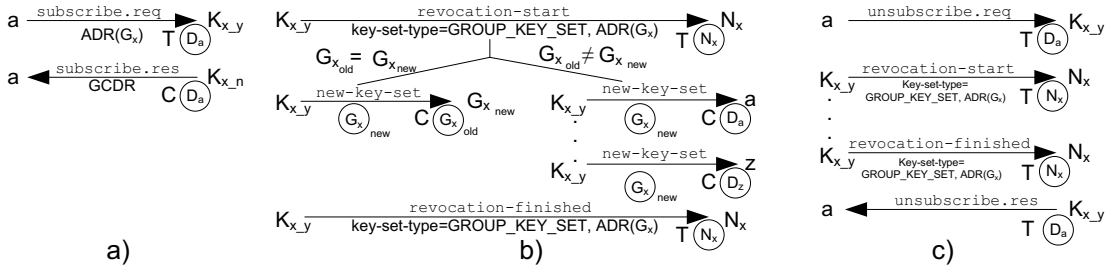

Figure 7. Group join (a), group key set revocation (b) and group leave (c)

network member so that each remaining network member can receive the key set. Although the approach eliminates the problem of multiple messages, the transmission of large messages contrasts BAN where protocols often exhibit only low bandwidth. Therefore, more advanced group key distribution schemes such as those surveyed in [17] need to be employed. As the choice also depends on the actual installation, a further analysis is not pursued.

At the end of the revocation process, the key server sends a `revocation-finished` message to all network members to inform them about the completion of the revocation process. This message is secured with the new network key set and triggers the mandatory use of the new network key set.

During the revocation process, a problem arises if the key server crashes after `revocation-start`. From this message on, the use of the old key set is prohibited but also the use of the new one is not foreseen until the key server sends the `revocation-finished` message. This problem can be solved with the help of key server clusters. Here, the other key servers within the cluster can monitor and, if necessary, restart the revocation process after a specific timeout has elapsed. Still, the value of this timeout is critical and has to be chosen with respect to the employed key distribution scheme.

In addition to joining a network, a device must be able to leave the network again in a secure manner (secure unbinding). To achieve this *network leave*, the device that wants to disconnect from the network sends a `network-leave.req` message (secured with a dynamic node key set) to one of the key servers. Then, the key server revokes the old network key and distributes a new one using the above mentioned mechanism. Finally, a confirmation is sent back to the device that left the network (`network-leave.res`).

4.2. Group key set management

Secure strict group communication requires all group members to share a group key set that is used to secure the exchanged data against malicious interference. Three different services are needed to manage these group key sets (cf. Figure 7). To retrieve a group key set, the device has to subscribe to the group (*group join*). A group join is similar to a network join. To start the join process, a device a sends a `subscribe.req` message to any key server within the key server cluster.[1] The key server then verifies whether the requesting device is allowed to join the group. If not, the key server responds with a negative `subscribe.res` message. If the device is allowed to join the group, the key server sends a positive `subscribe.res` message. This message includes a so called *Group Configuration Data Record (GCDR)* which contains information on the group (e.g., group size), and most important the group key set which is used to communicate securely within the group.

The mechanism to revoke a group key set is similar to the one for revocation of a network key set (cf. Section 4.1). The only difference concerns the `key-set-type` parameter of the `revocation-start` request, which is now set to `GROUP_KEY_SET`. Additionally, the group identifier is also included into the message to identify which group key set shall be revoked.

To leave a group again, the same mechanism as for a network leave is used. Corresponding to a network leave, a group leave uses the services `unsubscribe.req` and `unsubscribe.res`.

4.3. Session key set management

Secure exchange of process data as well as management data using the client/server paradigm is accomplished during sessions (cf. Figure 8). Consider, for example, a device a wants to establish a secure session to a device b. To start the session, a sends a `session-start.req` message to one of the key servers within the key server cluster. The request contains the address of b and is secured using a dynamic node key set of a. After having received this request, the key server generates a session key set which is distributed to b using a confidential `session-init.req` message. The message contains the session key set as well as the address of a and is secured with the dynamic node key set of b. If b is capable of accepting the session establishment request of a (e.g., b has sufficient resources to host more sessions), b sends a `session-init.res` back to the key server. The key server is now informed that b will accept the session request of a. Thus, it distributes the session key set to a using a confidential `session-start.res` (secured

[1] To identify a key server, the device has two options: it can perform a network join or it knows the list of key servers a priori.

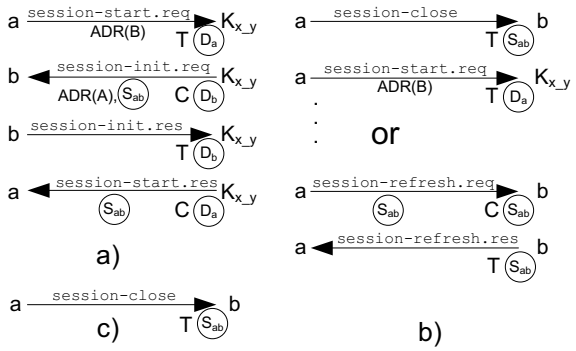

Figure 8. Session start (a), session key set revocation (b) and session destroy (c)

using the dynamic node key set of a). After reception of the session key set, a and b can communicate securely.

A session key set is valid only until one session partner decides to close the session. This is done by sending a trusted `session-close` message (secured using the session key set) to the session partner.

A session key set can be revoked in two different ways. First, a session partner can restart the session by closing the old session (using the `session-close` message) and starting a new one. This approach can be used if a session key has been compromised and a new one can thus not be secured using the old key set. Second, it is possible to generate a new session key set and distribute it during the currently active session (also called *session refresh*). This is done by sending a confidential `session-refresh.req` message to the session partner. The message contains the newly created session key set and is secured with the old one. The session partner responds with a `session-refresh.res` message which acknowledges the successful retrieval of the new session key set.

5. Key server distribution

As indicated in Section 3.2, a two-tiered key server infrastructure is used in the proposed solution: First, each network segment has a dedicated key server cluster. Second, each of these key server clusters consists of one or more key servers. To avoid inconsistent key sets, two possible solutions exist. On the one hand, the key set space can be split into equal parts with the key sets being distributed across the different key servers (*key set distribution*). The main advantage is that a failure of a single key server only effects a smaller number of key sets. However, the main drawback is that the single point of failure remains. If a key server fails, all key sets that are assigned to this key server become unavailable.

On the other hand, a redundancy scheme where each key set is replicated to all key servers can be employed (*key set replication*). Using such a scheme, the failure of a single key server does not effect the availability of the key sets since a device can simply contact another key server. The price for this redundancy is the synchronization effort, as any changes must be propagated to all other key servers to avoid inconsistencies.

The main aim of the proposed solution is to combine the advantages of both schemes. First, the key set space is split and distributed across the network segments. This means that each key server cluster is responsible for a part of all key sets. Second, these key sets are replicated among all key servers within a key server cluster.

In the following, this hybrid solution is discussed separately for each key set type:

- *Node key set*: Each key server cluster only stores the node key sets of the corresponding network segment's members. Within the key server cluster, these node key sets are replicated among all key servers. Using this scheme, each device can securely communicate with all key servers of its network segment. The node key sets have to be uploaded during installation in a secure manner. Guidelines how such an initial node key set distribution can be performed are described in [2].
- *Dynamic node key set*: Since dynamic node key sets are derived from the node key sets, each key server cluster is only responsible for the dynamic node key sets of its corresponding network segment. As dynamic node key sets are only valid during a single key set management request, their replication is not necessary.
- *Network key set*: Each key server cluster is responsible for maintaining the network key set of its network segment. Within the key server cluster, the network key set is replicated.
- *Group key set*: In this paper, it is assumed that exactly one key server cluster is responsible for a communication group. This maintaining key server cluster acts as a group coordinator and is responsible for managing the group membership and the group key set of the corresponding communication group. The assignment scheme depends on the underlying technology. For example, the assignment of the communication group to its maintaining key server cluster can be based on the amount of group members within the network segment in which the cluster is located. Alternatively, a special group address assignment where each group address is uniquely mapped to a network segment can be used. Definition of such an assignment scheme is left open to implementation. Within the key server cluster, all group key sets the key server cluster is responsible for are replicated.
- *Session key set*: Each key server cluster is responsible for handling session establishment requests within its network segment. Since a session key set is only valid during a single session, a replication of session key sets is not necessary.

Following this approach, each device can freely choose any key server within the key server cluster without get-

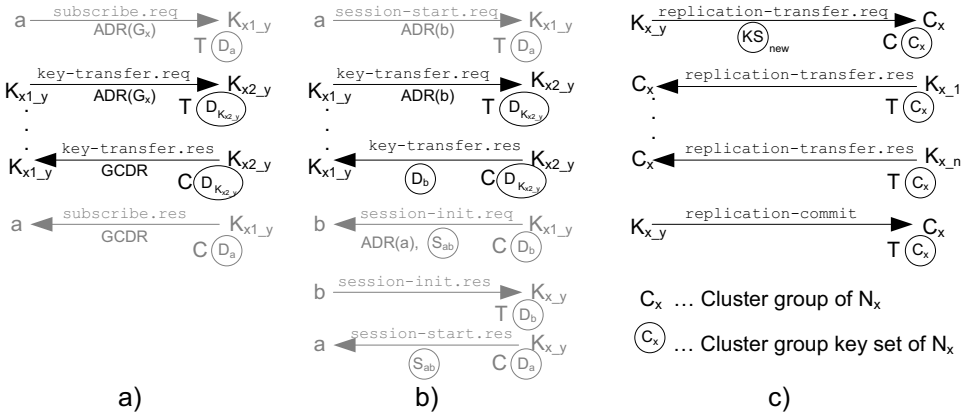

Figure 9. Group key set miss (a), node key set miss (b) and key set replication(c)

ting inconsistent key sets. However, there are situations in which a key set management request cannot be immediately satisfied by the key server. A possible solution to such situations is discussed in Section 5.1. Furthermore, replication requires a thought-out synchronization mechanism among the key servers within the key server cluster. This mechanism is described in Section 5.2.

5.1. Key forwarding

There are three situations in which a key set retrieval cannot be immediately satisfied by a key server. First, it is possible that two communication partners located in different network segments want to establish a session. In this case, the key server contacted by the session initiator does not hold the node key set of the second communication partner (*node key set miss*). Thus, it cannot distribute the generated session key set to the second communication partner (cf. Section 4.3). Second, it is possible that a device wants to join a group the key server cluster is not responsible for. In such a case the key server does not possess the GCDR and so it has to obtain it from the maintaining key server cluster (*group key set miss*). Third, it is possible that the members of a communication group that uses secure loose group communication are located in different network segments. Since the communication is secured using the network key set, only the group members that are located at the sender's network segment are able to handle these messages (*network key set conflict*).

To overcome a network key set conflict, the routers of the network segment have to forward all loose group communication messages. This means that the router has to disassemble the message using the network key set of the incoming network segment, generate a secured message using the network key set of the target network segment and forward it.[2] However, to avoid that routers see the clear text of forwarded messages, they can simply forward the secured message to a key server of the destination network. In that case, this key server is responsible for reassembling the message using the destination network key set and broadcasting it. Clearly, the key server must also possess the source network key set for decryption.

To resolve a node key set miss and a group key set miss, a mechanism has to be provided that allows an exchange of key sets between key server clusters. The proposed mechanism is based on an extended variant of the key forwarding mechanism described in [2]. To achieve a key set exchange between key servers located in different key server clusters, each member of the key server cluster must be aware of the device addresses as well as the node key sets of its parent as well as of all its child key servers.

To retrieve a key set from a key server located in a neighbor key server cluster, the key server has to determine whether the request has to be directed to a key server within its parent or within one of its child clusters. This can for example be accomplished using a hierarchical addressing scheme or with the help of static tables. It is assumed that a key server is able to determine which of its neighbor key server clusters it has to contact next. The key set exchange is done by sending a trusted `key-transfer.req` message to any key server part of the desired key server cluster. The request itself is secured using a dynamic key set of the destination key server. If the destination key server has the desired key set, it transfers it using a confidential `key-transfer.res` response (again secured with the dynamic node key set). If the destination key server is not in possession of the desired key set, it also has to request it from the next cluster.

Using this key set forwarding mechanism, a group key set miss can be resolved by requesting the desired GCDR (cf. Figure 9a). However, due to security reasons, the exchange of a node key set is not recommended in case of a node key set miss. Instead, it is more secure to transfer a dynamic node key set, which is only valid for a single key set retrieval (cf. Figure 9b).

[2] It is assumed that the router has sufficient routing information to forward group messages to the desired network segments. However, forwarding the group messages to all connected network segments (except the incoming one) is also possible since the assumed tree-structure prevents loops.

5.2. Key set replication

As mentioned before, a mechanism that allows to synchronize the key sets within the key cluster is necessary. To achieve such a *key set replication*, each member of the cluster is member of a special communication group called *cluster group*. The corresponding group key set is called *cluster key set*. To initiate a key set replication, the key server that wants to replicate a new key set sends a confidential `replication-transfer.req` message to its cluster group (cf. Figure 9c). The request includes the new key set that shall be replicated and is secured using the cluster key set. This request also indicates that the corresponding key set is locked. During this state, a simultaneous replication as well as a distribution to any device is not allowed. If a key server is ready for the replication, it replies with a positive `replication-transfer.res` message to the cluster group. Otherwise (e.g., it has already started a replication of the same key set), a negative acknowledgment is sent. However, if a key server does not respond at all, the initiator retransmits the request. After a defined number of retries, it must be assumed that the key server has crashed.

In case of a negative `replication-transfer.res` message, the replication process has to be aborted by the initiator. This is accomplished by sending a `replication-abort` message to the cluster group. If only positive acknowledgments are received (even if a key server crash is detected), the initiator finishes the replication by sending a `replication-commit` message to the other members of the cluster. Upon reception of the commit message, each key server replaces the old key set by the new one.

A problem arises if the initiator crashes after it sends `replication-transfer.req` since the other key servers will reside in state "locking". This problem can be solved by implementing a timeout mechanism. If a key server receives neither a `replication-abort` nor a `replication-commit` message, it decides to abort the replication process.

The replication protocol discussed above works as long as the key servers are assumed to be fail-silent. In a non fail-silent environment situations may occur in which the replication protocol leads to an inconsistent state. Consider, for example, the case of a key server that has missed a preceding key set replication and therefore distributes out-dated key sets. This situation can be solved by adding key revision numbers to the secured messages.

6. Conclusion

Given the technological developments in BAS, security can no longer be neglected. Our architecture is tailored to the resource constraints of embedded devices typically found in the building automation domain. It supports different data security classes and key set types. Unlike any other open protocol security approach, interaction between multiple key servers was given consideration.

Evaluating and judging the defined management services for retrieval and revocation of key sets raises the question of performance analysis under realistic operating conditions. To study the real-world behavior of our solution, we are finalizing a proof-of-concept implementation for a MSP430 architecture. To analyze the dynamical behavior especially in large installations, a simulation framework based on OMNeT++ [18] is underway.

References

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[2] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "Security in Networked Building Automation Systems", in *Proc. 6th IEEE Int. Workshop on Factory Communication Systems*, June 2006, pp. 283–292.

[3] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture", in *Proc. of the 6th International Conference on Information Processing in Sensor Networks*, 2007, pp. 479–488.

[4] M. Bellare, J. Kilian, and P. Rogaway, "The Security of Cipher Block Chaining", *Lecture Notes in Computer Science*, vol. 839, pp. 341–358, 1994.

[5] J. Song, R. Poovendran, J. Lee, and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, 2006.

[6] "The Keyed-Hash Message Authentication Code (HMAC)", FIPS PUB 198, National Institute of Standards and Technology, 2002.

[7] N. Okabe, S. Sakane, K. Miyazawa, K. Kamada, A. Inoue, and M. Ishiyama, "Security Architecture for Control Networks using IPsec and KINK", in *Proc. Symposium on Applications and the Internet (SAINT)*, 2005, pp. 414–420.

[8] "Advanced Encryption Standard (AES)", FIPS PUB 197, National Institute of Standards and Technology, 2001.

[9] P. Rogaway, M. Bellare, and J. Black, "OCB: A block-cipher mode of operation for efficient authenticated encryption", *ACM Transactions on Information and System Security*, vol. 6, no. 3, pp. 365–403, August 2003.

[10] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, 2003.

[11] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, 2005.

[12] "Control Network Protocol Specification", ANSI/EIA/CEA 709.1, 1999.

[13] "Information Technology - Home Electronic Systems (HES) Architecture", ISO/IEC 14543-3, 2006.

[14] "BACnet – A Data Communication Protocol for Building Automation and Control Networks", ANSI/ASHRAE 135, 2004.

[15] "BSR/ASHRAE Addendum g to ANSI/ASHRAE Standard 135-2004, BACnet – A Data Communication Protocol for Building Automation and Control Networks", April 2007, Second Public Review completed.

[16] ZigBee Alliance, "ZigBee Specification", 2006.

[17] S. Rafaeli and D. Hutchison, "A Survey of Key Management for Secure Group Communication", *ACM Computing Surveys*, vol. 35, pp. 309–329, 2003.

[18] A. Varga, "The OMNeT++ Discrete Event Simulation System", in *Proc. of the European Simulation Multiconference (ESM'2001)*, 2001.