6LoWPAN Security: Adding Compromise Resilience to the 802.15.4 Security Sublayer

Konrad-Felix Krentz
k-f.k@t-online.de

Hosnieh Rafiee
rafiee@hpi.uni-potsdam.de

Christoph Meinel
meinel@hpi.uni-potsdam.de

Hasso-Plattner-Institut, P.O. Box 900460, 14440 Potsdam, Germany

ABSTRACT

6LoWPAN is a protocol stack for seamlessly integrating 802.15.4-based wireless sensor networks with IPv6 networks. The security of 6LoWPAN widely depends on the 802.15.4 security sublayer. This sublayer also supports pairwise keys so as to mitigate node compromises. Currently, the establishment of pairwise keys is however unspecified. Moreover, broadcast keys are shared among multiple nodes, which is not compromise resilient. In this paper, we propose two energy-efficient and DoS-resilient 802.15.4 add-ons to fill these gaps: First, a pairwise key establishment scheme, which is adaptable to different 6LoWPAN networks and threat models. Second, an easy-to-implement and compromise-resilient protocol for authenticating broadcast frames. Together, our add-ons contain the effects of node compromises and provide a basis for detecting compromised nodes autonomously. We implemented both add-ons in Contiki and tested them on TelosB motes.

Categories and Subject Descriptors

C.2.1. [**Network Architecture and Design**]: Wireless communication; D.4.6. [**Security and Protection**]: Cryptographic controls

Keywords

Internet of things, sensor network security, link layer security, key management, broadcast authentication, DoS attacks.

1. INTRODUCTION

6LoWPAN stands for “IPv6 over Low-power Wireless Personal Area Networks”. Low-power Wireless Personal Area Networks (LoWPANs) are much like wireless sensor networks (WSNs). Both consist of battery-powered and computationally-restricted nodes. The deciding difference between them is that LoWPANs use the 802.15.4[1] radio standard per definition, whereas WSNs use any wireless communication standard. 6LoWPAN, in turn, refers to a protocol stack, which enables LoWPAN nodes, or nodes for short, to communicate with each other or remote hosts using IPv6. 6LoWPAN is envisioned to be adopted in diverse application areas, such as smart cities, industrial monitoring, and precision agriculture [19].

Many 6LoWPAN protocols[15, 31, 26] depend on the 802.15.4 security sublayer to filter out injected or replayed 802.15.4 frames. The 802.15.4 security sublayer serves this purpose by adding both a Message Integrity Code (MIC) and a frame counter to each frame. Currently, the establishment of 802.15.4 keys is however unspecified.

Preloading each node with a network-wide shared key is not a good solution. This is because 6LoWPAN networks run unattended in hostile environments, which makes these networks susceptible to node compromises. In a node compromise attack, an attacker physically tampers with a node so as to extract its cryptographic material. Tamper-resistant hardware is often proposed as a preventive measure [5], but is expensive and may still be vulnerable[4]. Once an attacker has obtained the network-wide key, the attacker can inject arbitrary frames anywhere at any time. Moreover, the attacker can then add unauthorized nodes to the victim 6LoWPAN network since upper-layer protocols[31, 26] rely on the 802.15.4 security sublayer.

Another straightforward scheme is the fully pairwise keys scheme, where each node is preloaded with a pairwise key for communication with any other node [9]. This is more compromise resilient, but three problems remain:

1. First, the fully pairwise keys scheme may be too memory-consuming for large-scale 6LoWPAN networks. For instance, when using 128-bit keys in a network of 32,768 nodes, each node has to store 500KB($\approx 32,767 \times 16$ bytes) of pairwise keys. This is already half of the total amount of external flash memory on a TelosB[23] mote - a typical node. Moreover, to support the addition of new nodes at runtime, each node needs to store a pairwise key for communication with each not-yet-deployed node in addition. Fewer pairwise keys can be preloaded if a node’s neighbors are known in advance, but this will complicate the deployment.

2. Second, there is a problem regarding the management of frame counters. To detect replayed frames, the most recent frame counter per source address needs to be stored. Deleting a frame counter when a node disappears is impossible because an attacker can then replay old frames. Therefore, over time, not all frame counters will fit in the limited random-access memory (RAM) on nodes, but some need to be swapped to external flash memory, which is energy consuming [23].

3. Third, while pairwise keys offer a compromise-resilient solution for securing unicast frames, there is no such solution for broadcast frames. Broadcast frames have to be authenticated with keys that are shared among neighboring nodes.

ASPI '13, September 08 2013, Zurich, Switzerland
Copyright 2013 ACM 978-1-4503-2543-1/13/09...$15.00.
http://dx.doi.org/10.1145/2523501.2523502

Thus, a node compromise not only reveals a single broadcast key, but also those of neighboring nodes. This is problematic when trying to detect compromised nodes because a malicious authentic broadcast frame can pretend to originate from an uncompromised node.

Unfortunately, public-key cryptography (PKC) is inappropriate for solving these problems. PKC remains, despite various optimizations[20], time and energy consuming on nodes. For example, an Elliptic Curve Digital Signature Algorithm (ECDSA) signature generation takes 3.2s and consumes 17.11mJ on a TelosB mote [20]. This may be acceptable for establishing a pairwise key with Elliptic Curve Menezes-Qu-Vanstone (ECMQV). However, attackers could exploit the relatively high energy consumption of ECDSA signature generations by repeatedly sending bogus key establishment requests. Consequently, attackers could deplete a node's battery. Moreover, ECMQV itself does not check whether a node is authorized to join a 6LoWPAN network. Hence, a certificate-based authorization mechanism should be used in addition. Such a mechanism would also be susceptible to denial-of-service (DoS) attacks since attackers could force nodes to verify the ECDSA signatures of bogus certificates. An ECDSA signature verification takes about 4s and consumes 21.82mJ on a TelosB mote [20]. Likewise, if ECDSA signatures were used to authenticate broadcast frames, this would also enable attackers to launch DoS attacks. Moreover, a software implementation of ECDSA consumes 28.2% of the program memory and 15% of the RAM on a TelosB mote [20].

We make two main contributions:

- We propose the Adaptable Pairwise Key Establishment Scheme (APKES). APKES is a framework for establishing pairwise 802.15.4 keys without PKC. Different pairwise key establishment schemes that forgo PKC can be plugged into APKES so as to adapt to different 6LoWPAN networks and threat models. This adaptability accounts for the current dilemma that indeed plenty of pairwise key establishment schemes that forgo PKC have been devised (see [9] for a survey), but none of them is universally applicable [3]. Furthermore, APKES features a much simpler approach to avoid the swapping of frame counters than was proposed in [21, 17].

- We propose the Easy Broadcast Encryption and Authentication Protocol (EBEAP). EBEAP is a compromise-resilient protocol for authenticating and encrypting 802.15.4 broadcast frames. Also EBEAP forgoes PKC. However, in contrast to existing broadcast authentication protocols that forgo PKC[22, 28], EBEAP needs neither delays, nor hash chains, nor time synchronization.

2. BACKGROUND

In this section, we provide background information for understanding the design of APKES and EBEAP. We will first outline the 6LoWPAN protocol stack and its vulnerabilities. Finally, we briefly explain the 802.15.4 security sublayer.

2.1 6LoWPAN and its Vulnerabilities

The 6LoWPAN protocol stack is shown in Figure 1. On Layer 1 and 2, the 802.15.4 media access control (MAC) and physical layer (PHY) transmit frames to one-hop neighbors. On Layer 2.5, the 6LoWPAN adaption layer[15] fragments and compresses IPv6 packets. Fragmentation is necessary because 802.15.4 has a Maximum Transmission Unit (MTU) of 127 bytes. Compression, on the other hand, reduces the energy consumption for transmitting and receiving IPv6 packets. On Layer 3, 6LoWPAN neighbor discovery (6LoWPAN-ND)[26] disseminates context information for compressing arbitrary IPv6 network prefixes. Apart from this, 6LoWPAN-ND is a multi-hop version of IPv6 neighbor discovery and IPv6 stateless autoconfiguration. Also on Layer 3, the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL)[31] routes IPv6 packets. On Layer 7, User Datagram Protocol (UDP)-based protocols, such as the Constrained Application Protocol (CoAP), are commonly employed.

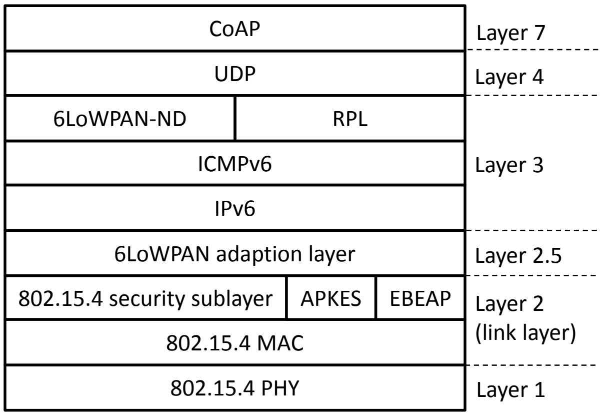

Figure 1: The 6LoWPAN protocol stack

Owing to the use of a wireless medium, attackers can inject and replay 802.15.4 frames. If the 802.15.4 security sublayer does not filter them out, such attacks can have severe consequences. On Layer 2.5, attackers can launch fragmentation attacks, which destroy partly reassembled IPv6 packets or exhaust buffers [16]. On Layer 3, an attacker can launch path-based DoS (PDoS)[10] attacks. In a PDoS attack, an attacker injects bogus IPv6 packets, which are then routed through the 6LoWPAN network, thus expending battery power. Another attack on Layer 3 is to inject bogus Internet Control Message Protocol for IPv6 (ICMPv6) messages so as to cripple RPL or 6LoWPAN-ND.

2.2 The 802.15.4 Security Sublayer

Figure 2 shows the format of a secured 802.15.4 frame. The essential security-related fields are a MIC and a frame counter. These fields enable receivers to verify both the authenticity and the freshness of a frame. The MICs are being generated with CCM* - a modified version of Counter with Cipher Block Chaining-MIC (CCM) [30]. CCM* in turn uses the Advanced Encryption Standard (AES) block cipher with 128-bit keys. The optional Key Identifier field can carry a reference to the key that was passed to CCM*.

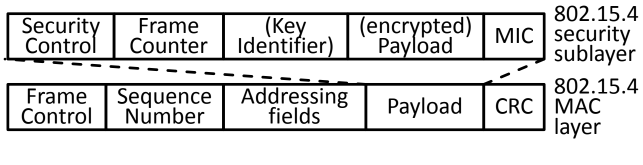

Figure 2: Format of a secured 802.15.4 frame

CCM* can also encrypt the payload. This makes sense if the final destination is just one hop away, as is the case in most 6LoWPAN-ND and RPL interactions. Conversely, if the final destination is multiple hops away, the hop-by-hop encryption of the 802.15.4 security sublayer is inappropriate because each (potentially compromised) intermediary can decrypt the payload. In such cases, the payload should be encrypted with IPsec[24] or another end-to-end security solution.

3. THREE PLUGGABLE SCHEMES AND THEIR TRADE-OFFS

In this section, we reconsider the dilemma that there is currently not "the" scheme for establishing pairwise 802.15.4 keys. We begin with defining criteria that should be fulfilled by such a scheme in general. Then, we present the "as is" state of three example schemes that can be plugged into APKES and evaluate them according to the defined criteria. Finally, we point out the trade-offs between these schemes. For reference, the appendix summarizes our notations.

3.1 The IOWEU Criteria

An ideal scheme for establishing pairwise 802.15.4 keys should meet the criteria inoculation, opaqueness, welcomingness, efficiency, and universality (IOWEU):

Inoculation: Inoculation was originally defined as the property that an attacker cannot "aid unauthorized nodes to join a network by compromising a small number of sensor nodes"[11]. However, most pairwise key establishment schemes cannot prevent so-called node replication attacks. In a node replication attack, an attacker reuses the address, or more generally speaking, the identity of a compromised node on an unauthorized node. Nevertheless, if a pairwise key establishment scheme at least guarantees that unauthorized nodes cannot use another address than that of a compromised node, concluding which node was compromised will be facilitated. Also, this will allow for blacklisting the address of a compromised node. Therefore, we relax the original definition of inoculation. We do not require an inoculated pairwise key establishment scheme to prevent node replication attacks.

Opaqueness: We also slightly change the original definition of opaqueness [11]. We call a pairwise key establishment scheme opaque if, in the event of node compromise, only links from and to the compromised node get compromised. Opaqueness is very useful when it comes to detecting malicious nodes. Detection will be difficult if an attacker not only gets the pairwise keys of links from and to compromised nodes, but also of links between uncompromised nodes. This is because a receiver of a malicious frame could not decide if the sender was compromised or not.

Welcomingness: A pairwise key establishment is welcoming if it supports the addition of new nodes at runtime.

Efficiency: Efficiency relates to the energy and memory efficiency of a pairwise key establishment scheme.

Universality: A pairwise key establishment scheme is universal if it does not assume a restrictive network model.

3.2 Localized Encryption and Authentication Protocol

The main idea of the Localized Encryption and Authentication Protocol (LEAP)[32] is to preload each node with a master key K_m and to erase K_m after the pairwise key establishment. K_m is used during the pairwise key establishment phase as follows. Suppose a node u is being deployed. u first derives its individual key K_u from K_m as $K_u = F(K_m, ID_u)$. Thereby, F is a pseudorandom function family, which is passed the seed K_m and the input ID_u. ID_u, on the other hand, is a unique identifier of u, such as u's address. Then, u broadcasts a HELLO message:

$$u \rightarrow * : \texttt{HELLO}\ \langle ID_u \rangle$$
$$v \rightarrow u : \texttt{ACK}\ \langle ID_v, ID_u \rangle_{K_v}$$

A neighbor v replies with an ACK message. The ACK message is authenticated with a MIC that is generated with K_v - the individual key of v. When u receives the ACK message from v, u still has K_m. Thus, u can generate K_v and hence verify the attached MIC. Once u and v discovered each other, both u and v calculate their pairwise key $K_{u,v}$ as $F(K_v, ID_u)$. Finally, after u has established a pairwise key with each of its neighbors, u erases K_m so that K_m will not leak in the event of a node compromise.

Inoculation and opaqueness: LEAP is not inoculated because v has no way to authenticate u. However, APKES fixes this problem by adding bidirectional authentication to LEAP, as was also proposed in [11]. That is, when running LEAP in conjunction with APKES, LEAP is inoculated and opaque, as long as the master key remains secret.

Welcomingness: LEAP is welcoming under the assumption that no jamming attacks occur during neighbor discovery. To ensure the successful addition of nodes, one can, e.g., eavesdrop on pairwise key establishment messages while deploying nodes.

Efficiency: LEAP is the most energy- and memory-efficient scheme that we are aware of. Only a single key needs to be preloaded and no other nodes need to be contacted during pairwise key establishment.

Universality: LEAP does not support mobile nodes.

3.3 Blom's Scheme

Blom's scheme[6] uses the parameters λ, n, and l. λ denotes the number of node compromises that are tolerable in Blom's scheme. n denotes the number of nodes in the network including not-yet-deployed nodes. l is the desired key length in bits. Given these three parameters, Blom's scheme initially chooses a prime power q that is large enough to accommodate keys of length l ($q \geq 2^l$), as well as n node identifiers ($q > n$). All operations in Blom's scheme will be done over the finite field $\mathbb{F}_q$. Next, a secret matrix $D \in \mathbb{F}_q^{(\lambda+1)\times(\lambda+1)}$ and a public matrix $G \in \mathbb{F}_q^{(\lambda+1)\times n}$ are generated randomly. The only requirements on D and G are that D is symmetric and that the columns of G are linearly independent. Then, the matrix $A = (DG)^T \in \mathbb{F}_q^{n\times(\lambda+1)}$ is being calculated. Finally, a new node u is preloaded with the ID_u-th row vector of A (denoted by $A_{ID_u,-}$) and the ID_u-th column vector of G (denoted by G_{-,ID_u}), where $1 \leq ID_u \leq n$.

Note that K is symmetric:

$$K = (DG)^T G = G^T D^T G = G^T DG = ((DG)^T G)^T = K^T$$

If two nodes u and v want to establish a pairwise key, they exchange their columns G_{-,ID_u} and G_{-,ID_v} and calculate their pairwise key as:

$$K_{ID_u,ID_v} = A_{ID_u,-}G_{-,ID_v} = A_{ID_v,-}G_{-,ID_u}$$

This procedure is depicted in Figure 3.

When choosing G^T as a Vandermonde matrix and q as a prime number, the exchange of the columns of G can be omitted and the pairwise key can be directly calculated as [12]:

$$K_{ID_u,ID_v} = \sum_{j=1}^{\lambda+1} A_{ID_u,j}(ID_v)^{j-1} = \sum_{j=1}^{\lambda+1} A_{ID_v,j}(ID_u)^{j-1}$$

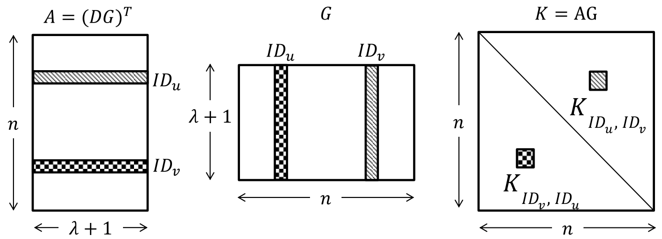

Figure 3: Blom's scheme

Thus, preloading the columns of G becomes omissible, too.

Inoculation and opaqueness: As long as no more than λ nodes are compromised, Blom's scheme is perfectly inoculated and opaque. Otherwise, the attacker obtains a system of linear equations with the $\lambda + 1$ variables of D. Thus, the attacker can calculate D and subsequently A.

Welcomingness: Blom's scheme is welcoming if the parameter n takes future deployments into account.

Efficiency: Both the computational effort and the memory consumption of Blom's scheme increase linearly with λ. Moreover, the row vector of A should be stored in the limited program memory or RAM on nodes since each generation of a pairwise key requires the entire row vector. An on-demand retrieval of this row vector from external flash memory would be energy consuming [23].

Universality: Blom's scheme is universal.

3.4 Random Pairwise Keys Scheme

Recall that in the fully pairwise keys scheme, each node is preloaded with a pairwise key for communication with any other node. By contrast, in the random pairwise keys scheme[8], only some pairs of nodes have pairwise keys. For example, in a network of $10,000$ nodes, each node has to store only 75 pairwise keys to achieve a probability of 0.5 of having a pairwise key with another node [14]. The reasoning is similar to that of the birthday paradox. Furthermore, if each node has at least 20 neighbors, the network is connected with probability 0.99999 [14]. Two nodes without a pairwise key have to employ an intermediary node for path key establishment. Thereby, the intermediary node unfortunately learns the established path key. To mitigate this problem, multiple intermediaries can be employed [8].

Inoculation: The random pairwise keys scheme is inoculated.

Opaqueness: In the random pairwise keys scheme, directly established pairwise keys are opaque, but path keys are revealed to intermediaries.

Welcomingness: The random pairwise keys scheme is welcoming.

Efficiency: The memory consumption of the random pairwise keys scheme is low. Furthermore, the retrieval of a pairwise key from external flash memory is not too energy consuming since a pairwise key is relatively short. The energy-consuming part is the path key establishment.

Universality: The random pairwise keys scheme requires a minimum network density.

3.5 The "IOWEU Dilemma"

The limiting factors of LEAP are universality, inoculation, and opaqueness. LEAP's universality is limited since LEAP does not support 6LoWPAN networks with mobile nodes. Moreover, LEAP should not be used if the secrecy of the master key cannot be guaranteed since inoculation and opaqueness are not ensured otherwise.

The limiting factor of Blom's scheme is efficiency. λ cannot be chosen arbitrarily high since both the memory consumption and the computational effort increase linearly with λ.

The limiting factors of the random pairwise keys scheme are opaqueness and universality. Its opaqueness is not perfect since path keys are not opaque. This complicates the detection of compromised nodes since a malicious frame could pretend to originate from an uncompromised node. If no full connectivity is needed, a drastic solution will be to skip the path key establishment and to therefore preload more pairwise keys. Its universality is also not perfect since the random pairwise keys scheme requires a minimum network density. This is problematic if a 6LoWPAN network has a deformed topology, such as when monitoring a pipeline.

In short, LEAP, Blom's scheme, as well as the random pairwise keys scheme require a trade-off between the IOWEU criteria. In terms of inoculation and opaqueness, the fully pairwise keys scheme remains the best choice and should be used whenever its memory consumption is acceptable.

4. APKES: ADAPTABLE PAIRWISE KEY ESTABLISHMENT SCHEME

The principle of APKES is as follows. The planner of a 6LoWPAN network picks an appropriate pairwise key establishment scheme and plugs it into the APKES implementation of each node. Table 1 defines four example schemes that can be plugged into APKES. The plugged-in scheme merely implements two functions, which provide APKES with shared secrets. However, the plugged-in scheme neither communicates with neighbors nor generates the actual pairwise keys. This is done by APKES. An exception is the random pairwise keys scheme, which may implement path key establishment in addition. Below, we first outline and then detail APKES. Finally, we analyze the security of APKES.

4.1 Protocol Overview

APKES involves three phases:

Optional preloading of short addresses: As described in the previous section, pluggable schemes usually need IDs for generating shared secrets. APKES reuses 802.15.4 addresses as IDs.

An 802.15.4 address consist of two parts. The first part is a 2-byte PAN-ID, which designates a subnetwork, or equivalently a Personal Area Network (PAN), of a LoWPAN. The second part is either a 2-byte short or an 8-byte extended address. Short addresses are only unique within a PAN. They can be preloaded manually or configured automatically with 6LoWPAN-ND. Extended addresses are globally-unique 64-bit Extended Unique Identifiers (EUI-64s). Each 802.15.4 transceiver has an EUI-64 assigned to it.

If APKES needs a shared secret with a node u, APKES passes the PAN-ID of u (denoted by PAN_u), the extended address of u (denoted by EA_u), and, if present, the short address of u (denoted by SA_u) to the plugged-in scheme. Some pluggable schemes may require the presence of short addresses. For example, in Blom's scheme, the dimensions of the matrices A and G will get too large otherwise. Unfortunately, the

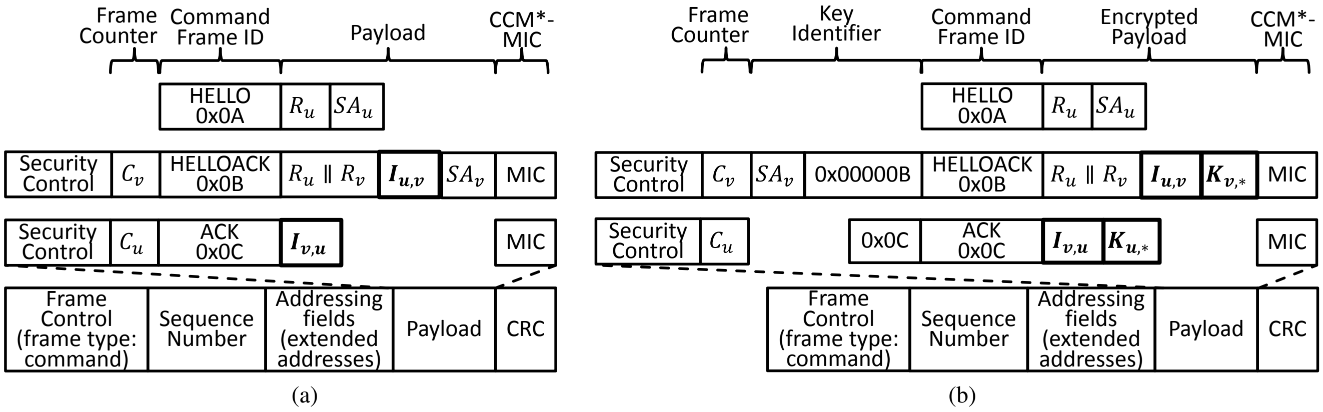

Figure 4: Format of APKES' commands (a) when broadcast encryption is off and (b) when broadcast encryption is on

Table 1: Shared secrets in different pluggable schemes

Pluggable scheme	Shared secret with `HELLO` sender u	Shared secret with `HELLOACK` sender v
Fully pairwise keys [9]	$K_{v,u}$	$K_{u,v}$
LEAP [32]	K_v	AES(K_m, EA_v) if K_m was not yet erased or else discard the `HELLOACK`
Blom's scheme [6, 12]	$\sum_{j=1}^{\lambda+1} A_{SA_v,j}(SA_u)^{j-1}$	$\sum_{j=1}^{\lambda+1} A_{SA_u,j}(SA_v)^{j-1}$
Random pairwise keys [8]	$K_{v,u}$ if present or else discard the `HELLO` command	$K_{u,v}$ if present or else discard the `HELLOACK`

automatic configuration of short addresses with 6LoWPAN-ND is incompatible with APKES. This is because 802.15.4 keys have to be established before ICMPv6 messages can be sent. Therefore, if the plugged-in scheme requires short addresses, these short addresses must be preloaded manually in this phase.

Preloading of cryptographic material: Prior to deploying a node u, u needs to be preloaded with its cryptographic material. APKES itself just needs a seed S_u for generating random numbers. In addition, u is preloaded with the specific cryptographic material of the plugged-in scheme. For example, when using LEAP, the master key K_m is preloaded in this phase.

Pairwise key establishment: The pairwise key establishment phase is initiated when pairwise keys with neighboring nodes are to be established. For doing so, APKES employs a three-way message exchange. The three messages are sent as three newly defined 802.15.4 commands, namely `HELLO`, `HELLOACK`, and `ACK` commands. Commands are special frames, which are not passed to an upper-layer protocol, but are processed at Layer 2. This enables APKES to establish pairwise 802.15.4 keys independently from upper-layer protocols.

Suppose a node u initiates the pairwise key establishment phase and discovers a neighbor v. Then, the protocol trace of the three-way handshake looks as follows:

$$\begin{aligned}
u &: \text{Generate } R_u \text{ randomly} \\
u \rightarrow * &: \texttt{HELLO}\ \langle R_u \rangle \\
v &: \text{Generate } R_v \text{ randomly and wait for } T_w \leq M_w \\
v &: K_{v,u} = \text{see Table 1} \\
v \rightarrow u &: \texttt{HELLOACK}\ \langle R_u, R_v \rangle_{K_{v,u}} \\
v &: K'_{v,u} = \text{AES}(K_{v,u}, R_u \| R_v) \\
u &: K_{u,v} = \text{see Table 1} \\
u &: K'_{u,v} = \text{AES}(K_{u,v}, R_u \| R_v) \\
u \rightarrow v &: \texttt{ACK}\ \langle\rangle_{K'_{u,v}}
\end{aligned}$$

Initially, u generates a random number R_u and broadcasts a `HELLO` command. The `HELLO` command contains R_u. Upon receipt of the `HELLO` command on v, v generates a random number R_v. Furthermore, v stores $R_u \| R_v$ and waits for a random waiting period T_w, which has a maximum of M_w. This prevents u from being overwhelmed with `HELLOACK`s. As soon as T_w elapses, v sends a `HELLOACK` containing both R_u and R_v. The `HELLOACK` is authenticated with $K_{v,u}$ as CCM* key. $K_{v,u}$ is the shared secret between v and u and is provided by the plugged-in scheme, as defined in Table 1. Furthermore, v derives the actual pairwise key $K'_{v,u}$ from $K_{v,u}$ as AES$(K_{v,u}, R_u \| R_v)$. Upon receipt of the `HELLOACK` on u, u verifies the attached CCM*-MIC by obtaining the shared secret $K_{u,v}$ from the plugged-in scheme. In addition, u checks whether the challenge R_u remained unchanged. When all these checks are successful, u derives the pairwise key $K'_{u,v}$, too. Finally, u sends an `ACK` to v, which is authenticated with $K'_{u,v}$. Upon receipt of the `ACK` on v, v verifies the attached CCM*-MIC by using the already-generated pairwise key $K'_{u,v}$.

4.2 Protocol Specification

Figure 4a shows the format of APKES' commands. Ignore the bold fields for the moment - their use will become apparent when we discuss EBEAP in the next section. Note that the addressing fields always contain extended addresses and that `HELLO` and `HELLOACK` commands contain the sender's short address in addition. As already mentioned, these addresses are passed to the

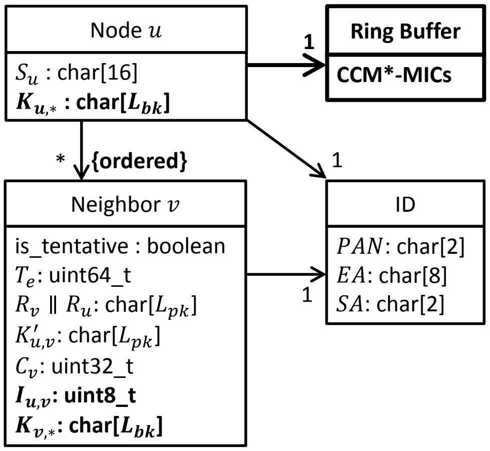

Figure 5: Data on a node u that runs both APKES and EBEAP

plugged-in when getting shared secrets. If a sender has no short address, the short address field is set to 0xFFFF. The first payload byte of each command is the command's identifier, as specified in the 802.15.4 standard. The proposed values 0x0A, 0x0B, and 0x0C are currently unused. The length of the random numbers $|R_u| = |R_v|$ is calculated as $\frac{L_{pk}}{2}$, where L_{pk} is the length of pairwise keys in bytes. L_{pk} is left configurable by APKES and may take the values $L_{pk} \in \{2, 4, \ldots, 16\}$. Choosing $L_{pk} < 16$ saves RAM, but is less secure. Additionally, HELLOACKs and ACKs carry the fields of the 802.15.4 security sublayer. Thereby, C_u denotes the 4-byte frame counter of a node u.

If APKES were implemented as described so far, attackers could launch DoS attacks on APKES by flooding HELLO commands. Receivers will always reply with a HELLOACK and hence expend their battery power. Likewise, replayed HELLOACKs trigger ACKs. To prevent these attacks, and to prevent injection and replay attacks in general, each node stores its neighbors as shown in Figure 5. There are two kinds of neighbors:

Tentative neighbors: Tentative neighbors are created upon receipt of a HELLO command if two conditions are met. First, the sender v must not already be stored as a neighbor. Second, the number of tentative neighbors of the receiver u must be smaller than M_t. Before checking the second condition, u deletes all expired tentative neighbors. A tentative neighbor expires at $T_e = T_c + T_w + T_a$, where T_c is the time when the tentative neighbor was created, T_w is the waiting period until the HELLOACK will be sent to the tentative neighbor, and T_a is the waiting period for the ACK from the tentative neighbor. Since the maximum number of tentative neighbors is restricted to M_t, multiple consecutive HELLO commands are being discarded. When a tentative neighbor is created, several fields are set, namely $R_v \| R_u$, is_tentative, T_e, SA_v, EA_v, PAN_v, and after the HELLOACK to v was sent, $K'_{u,v}$.

Permanent neighbors: Permanent neighbors are potentially created upon receipt of valid HELLOACKs and ACKs. Upon receipt of a valid HELLOACK, there are three cases. First, if the sender v is not already a neighbor, v is stored as a permanent neighbor. Second, the sender v may already be a permanent neighbor. This can happen when an ACK gets lost or when the received HELLOACK was replayed. In this case, the receiver u only sends another ACK if the stored frame counter C_v of v is smaller than the frame counter within the received HELLOACK from v. If so, u updates all information about v, especially $K'_{u,v}$ since R_v will have changed. Third, if the sender v is a tentative neighbor, v is turned into a permanent neighbor and an ACK is sent. This avoids a deadlock situation when multiple nodes perform pairwise key establishment in parallel. Upon receipt of a valid ACK from v, v is only turned into a permanent neighbor if v is currently a tentative neighbor. In brief, permanent neighbors are created upon receipt of valid nonreplayed HELLOACKs and upon receipt of valid ACKs from nontentative neighbors. When a permanent neighbor is created, only T_e, as well as the random numbers $R_v \| R_u$ remain unset.

A disappeared permanent neighbor should be deleted automatically like in 6LoWPAN-ND, but that is beyond the scope of this paper.

APKES modifies the 802.15.4 security sublayer to discard frames from nonpermanent neighbors right away. Only frames from a permanent neighbor v are decrypted, verified, and (if authentic and fresh) passed to the 6LoWPAN adaption layer. If so, C_v is thereby updated.

4.3 Security Analysis

APKES mitigates HELLO flood attacks by limiting the number of tentative neighbors. To reinforce this mechanism, the maximum number of tentative neighbors M_t should be low and the waiting period for an ACK T_a should be long. Unfortunately, an attacker can also create tentative neighbors and thus prevent the pairwise key establishment. Another approach would have been to accept unlimited amounts of tentative neighbors, e.g., through a SYN cookie-like mechanism. In this case, however, each HELLO command would still trigger a HELLOACK.

Furthermore, APKES prevents the replay of HELLOACKs and ACKs. The replay of HELLOACKs is prevented directly through frame counters. Also the replay of HELLOACKs from deleted neighbors is impossible since the expected random number R_u will have changed. The replay of an ACK is senseless since a tentative neighbor is turned into a permanent neighbor just once. The situation is different when a permanent neighbor was deleted. Then, an attacker could try to become a permanent neighbor by first spoofing a HELLO command and second replaying a corresponding ACK. However, the random number R_v will have changed and therefore $K'_{v,u} = \text{AES}(K_{v,u}, R_u \| R_v)$, too. Thus, the CCM*-MIC of the replayed ACK will not be accepted.

In Table 1, we defined that individual keys in LEAP are generated using AES. This is practical since AES is required for implementing CCM* anyway. The security of this definition depends on the pseudorandomness, as well as the resistance against key recovery attacks of AES.

Likewise, APKES uses AES for deriving the actual pairwise keys. This has two advantages. First, disappeared permanent neighbors can be deleted along with their frame counters. This is thanks to the random numbers R_u and R_v, which cause a different pairwise key to be established when a neighbor reappears. Hence, replayed frames from previous interactions will be detected and discarded. Second, this improves the opaqueness of some pluggable schemes. For example, in LEAP, the attacker not only needs to obtain the master key to derive the pairwise key between two nodes, but also to eavesdrop on the random numbers R_u and R_v. Similarly, this mechanism improves the opaqueness of Blom's scheme. Compromising $\lambda + 1$ nodes is not sufficient to obtain all pairwise keys. Also the random numbers are needed. Since $|R_u| + |R_v| = L_{pk}$, guessing R_u and R_v is as difficult as guessing a pairwise key.

5. EBEAP: EASY BROADCAST ENCRYPTION AND AUTHENTICATION PROTOCOL

While APKES enables nodes to authenticate and encrypt unicast frames, EBEAP enables nodes to authenticate and encrypt broadcast frames. Broadcast frames are, e.g., used in RPL and 6LoWPAN-ND to discover neighboring nodes and to inform neighboring nodes about network changes in an efficient manner. As already discussed, authenticating broadcast frames with ECDSA signatures would enable DoS attacks. Instead, EBEAP solely uses CCM* and reuses the established pairwise keys of APKES. As in the previous section, we will first outline and then detail EBEAP. Finally, we analyze the security of EBEAP.

5.1 Protocol Overview

Let $M(k, f)$ denote the CCM*-MIC over a frame f with key k. Suppose a node u wants to send a broadcast frame f to its permanent neighbors $v_0, v_1, \ldots, v_{n-1}$. In this case, u first adds a Security Control, as well as a Frame Counter field to f to obtain f'. Then, u broadcasts two frames:

$$u \rightarrow * : \texttt{ANNOUNCE}\ \langle M(K'_{u,v_0}, f')\|...\|M(K'_{u,v_{n-1}}, f')\rangle$$
$$u \rightarrow * : f'$$

The first broadcast is an ANNOUNCE command, which contains one CCM*-MIC over f' for each permanent neighbor v_i. Thereby, K'_{u,v_i} is the established pairwise key from APKES. Note that the ordering of the CCM*-MICs corresponds to the ordering of the neighbor list of u. Upon receipt of the ANNOUNCE command on a permanent neighbor v_i of u, v_i extracts and stores its designated CCM*-MIC in a ring buffer, which automatically overwrites old CCM*-MICs. However, to enable v_i to extract its designated CCM*-MIC, v_i needs its index $i = I_{v,u}$ in the neighbor list of u. These indices are piggybacked on APKES' commands, as will be explained in the next subsection. The second broadcast is the actual broadcast message f'. Upon receipt of f' on v_i, v_i generates $M(K'_{v_i,u}, f')$. If the resulting CCM*-MIC is stored in v_i's ring buffer and if f' is fresh, v accepts f'.

Optionally, EBEAP also supports the encryption of broadcast frames. If this feature is on, each node u generates a broadcast key $K_{u,*}$ and passes it securely to its neighbors. We will discuss how this is done in the next subsection. Broadcast keys are not used for authenticating f', but only for encrypting f'. When encryption is on, EBEAP's protocol trace looks as follows:

$$u \rightarrow * : \texttt{ANNOUNCE}\ \langle M(K'_{u,v_0}, f')\|...\|M(K'_{u,v_{n-1}}, f')\rangle$$
$$u \rightarrow * : \{f'\}_{K_{u,*}}$$

, where $\{f'\}_{K_{u,*}}$ denotes that the payload of f' is being encrypted with $K_{u,*}$. We will discuss the security implications later in the course of our security analysis of EBEAP.

5.2 Protocol Specification

EBEAP needs to store additional data on each node u, as marked in bold in Figure 5. In particular, u stores its index $I_{u,v} \in \{0, 1, \ldots\}$ in the neighbor list of each permanent neighbor v. Furthermore, u now has a ring buffer for storing received CCM*-MICs. Broadcast keys are only stored in addition if broadcast encryption is on. If so, u generates its own broadcast key $K_{u,*}$ randomly by using its seed S_u. The length of broadcast keys is denoted by L_{bk} and is 16 bytes maximum.

Figure 4a and b show the format of APKES' commands when broadcast encryption is off and on, respectively. HELLOACKs and ACKs now piggyback the indices $I_{v,u}$ and $I_{u,v}$, respectively. Furthermore, when using broadcast encryption, HELLOACKs and ACKs piggyback the broadcast keys $K_{v,*}$ and $K_{u,*}$, respectively. However, the broadcast keys cannot be sent in plain text. Therefore, when using broadcast encryption, HELLOACKs and ACKs are not only authenticated, but also encrypted with CCM*. This incurs the problem that CCM* encrypts the whole payload. Thus, receivers could not decide which key to use for decrypting the payload since they cannot read either the contained command frame identifier or short address. Therefore, when using broadcast encryption, this data is put within the Key Identifier field, which remains unencrypted.

Figure 6 shows the format of ANNOUNCE commands. Recall that 802.15.4 has an MTU of 127 bytes. To support the case where the CCM*-MICs do not fit within a single ANNOUNCE command, we added the First Index field. Its value is the index of the neighbor for whom the first CCM*-MIC is designated. This allows for sending multiple ANNOUNCE commands when a single ANNOUNCE command is insufficient. For example, when using 7-byte CCM*-MICs, no more than $15 = \lfloor \frac{127-21}{7} \rfloor$ CCM*-MICs fit within a single ANNOUNCE command.

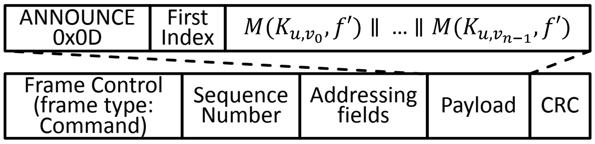
ANNOUNCE 0x0D
First Index
$M(K_{u,v_0}, f')$ ∥ … ∥ $M(K_{u,v_{n-1}}, f')$
Frame Control (frame type: Command)
Sequence Number
Addressing fields
Payload
CRC

Figure 6: Format of ANNOUNCE commands

Over time, expired tentative neighbors or deleted permanent neighbors can cause gaps in the neighbor list. To keep ANNOUNCE commands as short as possible, the neighbor list should never have gaps. To support the reordering of neighbors, we define another command: UPDATE. The format of UPDATE commands corresponds to that of HELLOACKs, except that the random numbers are left out. Furthermore, in contrast to HELLOACKs, UPDATE commands are secured with $K'_{v,u}$. The semantic of an UPDATE command is to replace the currently stored values of PAN_v, EA_v, SA_v, $I_{u,v}$, and $K_{v,*}$ with the values contained in the UPDATE command. UPDATE commands can also be used when a short address was obtained via 6LoWPAN-ND. In this case, neighboring nodes can be notified via UPDATE commands of the obtained short address.

5.3 Security Analysis

Let us consider the probability that a random broadcast frame gets accepted by a receiver. The frame will potentially be accepted by one of the n permanent neighbors of the sender. All other nodes will immediately discard the broadcast frame. Suppose each of the n permanent neighbors buffers a maximum of m CCM*-MICs. A single neighbor will hence accept the frame with probability $\frac{m}{2^l}$, where l is the length of CCM*-MICs in bits. Thus, the probability that any neighbor will accept the broadcast is $\frac{mn}{2^l}$. This is higher than the probability of accepting a random unicast, which is $\frac{1}{2^l}$. To compensate for this effect, our implementation allows users to configure the length of CCM*-MICs within ANNOUNCE commands and unicast frames independently. It is usually sufficient to use one additional byte per CCM*-MIC within ANNOUNCE commands. For example, when $m = 10$ and $n = 15$, this yields:

$$\frac{10 * 15}{2^{l+8}} < \frac{2^8}{2^{l+8}} = \frac{1}{2^l}$$

Since ANNOUNCE commands are sent unauthenticated, attackers can spoof them. Thus, before u sends the actual broadcast f', an attacker could overwrite the announced CCM*-MICs from u. If the attacker succeeds, u's permanent neighbors discard f'. However, since the permanent neighbors always overwrite the oldest CCM*-MIC first, the attacker needs to send multiple `ANNOUNCE` commands. Hence, choosing m very high prevents this attack, but entails the problem that the probability of accepting a random broadcast frame increases. Therefore, choosing m very high is not an option. Another mitigation technique is to authenticate `ANNOUNCE` commands with broadcast keys. The attached CCM*-MICs could be short since our goal is just to discard most of the spoofed `ANNOUNCE` commands. However, we think that this attack will be difficult anyway if all CCM*-MICs fit within a single `ANNOUNCE` command. This is due to the fact that the two broadcast frames are sent without delay. If multiple `ANNOUNCE` commands have to be sent, this attack becomes easier and puts a limit on the scalability of EBEAP.

If an attacker compromises a node, the broadcast keys of the node's permanent neighbors will leak. Thus, the above mitigation technique would no longer work. Moreover, the attacker can then decrypt the neighbors' broadcast frames. However, the attacker cannot get broadcast frames accepted that pretend to originate from one of the neighbors. This is due to the opaqueness of the pairwise keys between other pairs of nodes. This property is valuable when it comes to detecting compromised nodes and is the advantage of EBEAP over authenticating broadcast frames with shared broadcast keys.

6. IMPLEMENTATION

We implemented APKES and EBEAP in the WSN operating system Contiki[1] and made our implementation publicly available[2]. Contiki's 6LoWPAN stack is organized like the theoretical 6LoWPAN protocol stack shown in Figure 1. However, Contiki currently does not allow for an additional layer in between the MAC and the 6LoWPAN adaption layer for implementing link layer security. We added such a layer, which enables us to plug-in arbitrary `llsec_drivers`. By default, the `nullsec_driver` is used, which merely passes incoming and outgoing frames to the upper and lower layer, respectively. Furthermore, we implemented the `coresec_driver`, which implements the 802.15.4 security sublayer, as well as our add-ons APKES and EBEAP. Thereby, we made three simplifications. First, we did not implement `UPDATE` commands. Second, we restricted the maximum number of neighbors so that a single `ANNOUNCE` command is always sufficient. Third, only LEAP is currently available as a pluggable scheme.

The CCM* implementation of the `coresec_driver` is encapsulated behind an interface. For our TelosB motes, we implemented a CCM* driver, which leverages the hardware security-based AES implementation of the CC2420[3] transceiver. (The CC2420 does also implement CCM* directly. However, according to our tests, the output is not 802.15.4 compliant.) The interface of CCM* drivers also provides direct access to AES. The `coresec_driver` uses AES not only for deriving pairwise keys, but also for generating random numbers. Specifically, the `coresec_driver` generates random numbers by calling AES with the seed S_u as key and a nonce i as plain text. The output is a 16-byte random number [11].

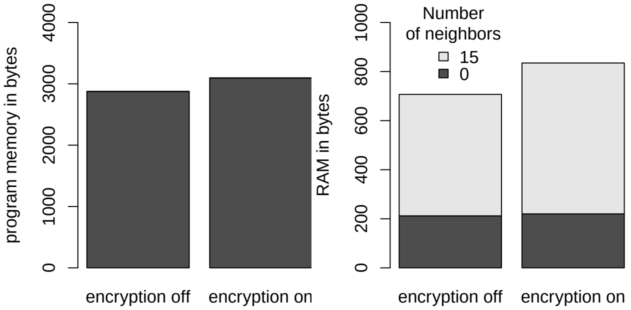

Figure 7: Memory footprint of the `coresec_driver`

7. EVALUATION

Figure 7 shows the memory footprint of the `coresec_ driver` on TelosB motes. We obtained these numbers with the tools `msp430-size` and `msp430-ram-usage`. The numbers comprise the implementation of CCM*, APKES, as well as EBEAP. The required program memory is acceptable on TelosB motes, which have 48KB of program memory. The RAM overhead increases linearly with the maximum number of neighbors. Our example configuration allows for 15 neighbors with 12-byte pairwise and 8-byte broadcast keys. The resulting RAM overhead is significant for TelosB motes, which have 10KB RAM. This shows that the ability to delete neighbors is crucial. For comparison, an ECDSA implementation alone consumes 13.5KB of program memory and 1.5KB of RAM on a TelosB mote [20].

We have not yet measured the energy consumption of the `coresec_driver`. However, Raza et al.[24] also leveraged the hardware security of the CC2420. According to their results, CCM* operations consume only a few microjoules. This indicates that the energy consumption for generating multiple CCM*-MICs, as done in EBEAP, is negligible, too. For this reason, we expect that the overhead in energy consumption of the `coresec_driver` is dominated by the transmission of additional security-related data. In particular, the transmission of the additional broadcast frames in EBEAP will be energy consuming. The energy consumption for transmitting frames highly depends on the employed MAC protocol. When using ContikiMAC[13], sending broadcast frames is relatively energy consuming since each broadcast frame is repeatedly transmitted during an entire wake-up interval. This consumes $\approx$1.8mJ on TelosB motes, irrespective of the broadcast's length. By contrast, a broadcast reception consumes only $\approx$0.2mJ in ContikiMAC. For comparison, an ECDSA signature generation or verification consumes 17.11mJ or 21.82mJ, respectively [20].

8. RELATED WORK

As an alternative to the 802.15.4 security sublayer, Hummen et al.[16] proposed mitigation mechanisms against known fragmentation attacks. Likewise, RPL[31] specifies mechanisms that can be used in lieu of the 802.15.4 security sublayer. However, these surgical security mechanisms introduce complexity and cannot prevent PDoS attacks. The 802.15.4 security sublayer protects against all these attacks at once and is hence more efficient. Beyond that, APKES prevents unauthorized nodes from joining a 6LoWPAN network.

The swapping of frame counters was avoided in [21, 17] by means of Bloom filters. Unfortunately, Bloom filters have a false positive rate, which causes legitimate frames to sometimes be discarded. Another approach is to use timestamps instead of counters [21], but this requires a secure time synchronization protocol [29].

[1] `http://contiki-os.org`

[2] `https://github.com/kkrentz/contiki/wiki`

[3] `http://www.ti.com/lit/ds/symlink/cc2420.pdf`

APKES avoids these drawbacks. The approach of APKES is to establish a different pairwise key when a neighbor reappears. This enables nodes to free up memory that was allocated for disappeared permanent neighbors and, in particular, their frame counters.

EBEAP bases on the Timed Efficient Stream Loss-tolerant Authentication++ (TESLA++)[28] protocol, which is in turn based on TESLA[22]. Like EBEAP, TESLA and TESLA++ use symmetric-key algorithms and send two broadcasts. In TESLA, the first broadcast is the actual broadcast message. Unfortunately, the receivers need to buffer the whole broadcast message until the second broadcast arrives. Moreover, the second broadcast must be delayed. This makes TESLA susceptible to memory-based DoS attacks. TESLA++ prevents such attacks by sending only the MIC of upcoming broadcast frames within the first broadcast like in EBEAP. However, TESLA++ still requires delaying the second broadcast. Another limitation of both TESLA and TESLA++ is their need for a secure time synchronization protocol[29] and hash chains. EBEAP needs neither hash chains, nor time synchronization, nor a delay in between the two broadcasts. Unfortunately, EBEAP does not scale well with the number of receivers due to additional CCM*-MICs.

Thus far, most efforts on link layer security in WSNs have focussed on performance[18, 7] and energy efficiency[21, 17], and have left pairwise key establishment unaddressed. An exception is L^3Sec[27], which integrates a variant of LEAP. Besides, two recent efforts[2, 25] use key distribution centers (KDCs) for bootstrapping 802.15.4 keys. However, KDCs are susceptible to PDoS attacks since bogus key establishment requests are routed all the way to the KDC. Instead, APKES establishes pairwise keys in a distributed fashion, which is PDoS resistant and more energy efficient. Furthermore, as opposed to the recent efforts[2, 25], APKES forgoes PKC.

9. CONCLUSIONS AND FUTURE WORK

Node compromises are a severe threat to 6LoWPAN networks. We have pointed out that inoculated and opaque pairwise key establishment schemes contain the effects of node compromises. Furthermore, we have presented APKES and EBEAP, which solve the three problems stated in the introduction in an energy-efficient, as well as DoS-resilient manner. Unfortunately, attackers can still inject malicious frames after compromising a node because they obtain its pairwise keys. However, APKES and EBEAP exhibit a nice property: If the plugged-in scheme is inoculated and opaque, the sender of a malicious authentic frame can be considered compromised. For future work, we will try to exploit this property to detect compromised nodes and to subsequently revoke their pairwise keys. We will also investigate whether the pairwise key establishment phase of APKES could be triggered self-adaptively through cooperation with RPL. At the moment, the `coresec_driver` initiates the pairwise key establishment phase only at start-up.

10. REFERENCES

[1] IEEE Standard 802.15.4, 2006. http://standards.ieee.org/.

[2] ZigBee IP Specification, 2013. http://www.zigbee.org.

[3] C. Alcaraz, J. Lopez, R. Roman, and H.-H. Chen. Selecting key management schemes for WSN applications. *Computers & Security*, 31(38):956–966, 2012.

[4] R. Anderson and M. Kuhn. Tamper resistance - a cautionary note. In *Proceedings of the Second USENIX Workshop on Electronic Commerce (WOEC'96)*, volume 2, pages 1–11. USENIX, 1996.

[5] A. Becher, Z. Benenson, and M. Dornseif. Tampering with motes: real-world physical attacks on wireless sensor networks. In *Security in Pervasive Computing*, pages 104–118. Springer, 2006.

[6] R. Blom. An optimal class of symmetric key generation systems. In *Advances in Cryptology - EUROCRYPT 84*, pages 335–338. Springer, 1984.

[7] L. Casado and P. Tsigas. ContikiSec: a secure network layer for wireless sensor networks under the Contiki operating system. In *Identity and Privacy in the Internet Age*, pages 133–147. Springer, 2009.

[8] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 197–213. IEEE, 2003.

[9] C.-Y. Chen and H.-C. Chao. A survey of key distribution in wireless sensor networks. *Security and Communication Networks*, 2011.

[10] J. Deng, R. Han, and S. Mishra. Defending against path-based DoS attacks in wireless sensor networks. In *Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '05)*, pages 89–96. ACM, 2005.

[11] J. Deng, C. Hartung, R. Han, and S. Mishra. A practical study of transitory master key establishment for wireless sensor networks. In *Proceedings of the First IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, pages 289 – 302. IEEE, 2005.

[12] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili. A pairwise key predistribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 42–51. ACM, 2003.

[13] A. Dunkels. The contikimac radio duty cycling protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[14] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)*, pages 41–47. ACM, 2002.

[15] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, 2011. Updates RFC 4944.

[16] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle. 6LoWPAN fragmentation attacks and mitigation mechanisms. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '13)*, pages 55–66. ACM, 2013.

[17] D. Jinwala, D. Patel, and K. Dasgupta. FlexiSec: a configurable link layer security architecture for wireless sensor networks. *Information Assurance and Security*, 4, 2012.

[18] C. Karlof, N. Sastry, and D. Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 162–175. ACM, 2004.

[19] E. Kim, D. Kaspar, and J. Vasseur. Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6568, 2012.

[20] A. Liu and P. Ning. TinyECC: a configurable library for elliptic curve cryptography in wireless sensor networks.

Technical Report TR-2007-36, North Carolina State University, 2008.

[21] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. MiniSec: a secure sensor network communication architecture. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, pages 479–488. ACM, 2007.

[22] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA broadcast authentication protocol. *RSA CryptoBytes*, 5(2), 2002.

[23] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005)*, pages 364–369. IEEE, 2005.

[24] S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt. Secure communication for the Internet of Things - a comparison of link-layer security and IPsec for 6LoWPAN. *Security and Communication Networks*, 2012.

[25] S. Raza, T. Voigt, and V. Juvik. Lightweight IKEv2: a key management solution for both compressed IPsec and IEEE 802.15.4 security. In *Proceedings of the IETF International Workshop on Smart Object Security*. IETF, 2012.

[26] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6775, 2012.

[27] H. Soroush, M. Salajegheh, and T. Dimitriou. Providing transparent security services to sensor networks. In *Proceedings of the IEEE International Conference on Communications (ICC '07)*, pages 3431–3436. IEEE, 2007.

[28] A. Studer, F. Bai, B. Bellur, and A. Perrig. Flexible, extensible, and efficient VANET authentication. *Journal of Communications and Networks*, 11(6):574–588, Dec. 2009.

[29] K. Sun, P. Ning, and C. Wang. TinySeRSync: secure and resilient time synchronization in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*, pages 264–277. ACM, 2006.

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

[31] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, 2012.

[32] S. Zhu, S. Setia, and S. Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 62–72. ACM, 2003.

APPENDIX

A. NOTATIONS

Symbol	Meaning
$u, v, \ldots$	nodes $u, v, \ldots$
$\langle \ldots \rangle$	insecure message or frame
$\langle \ldots \rangle_k$	message or frame that is authenticated with the key k
$\{\ldots\}_k$	frame whose payload is encrypted with the key k
$u \rightarrow v$	u sends a message to v
$u \rightarrow *$	u broadcasts a message
ID_u	unique identifier of u (e.g., its address)
AES(k, m)	single AES encryption of the plain text block m with the key k
A^T	transpose of the matrix A
$\|\|$	concatenation operator
$\|m\|$	length of m in bytes