Key Management Building Blocks for Wireless Sensor Networks

Yee Wei Law[†], Jeroen Doumen[‡] and Marimuthu Palaniswami[†]
[†]The University of Melbourne, Australia
[‡]University of Twente, The Netherlands

ABSTRACT

Cryptography is the means to ensure data confidentiality, integrity and authentication in wireless sensor networks (WSNs). To use cryptography effectively however, the cryptographic keys need to be managed properly. First of all, the necessary keys need to be distributed to the nodes before the nodes are deployed in the field, in such a way that any two or more nodes that need to communicate securely can establish a session key. Then, the session keys need to be refreshed from time to time to prevent birthday attacks. Finally, in case any of the nodes is found to be compromised, the key ring of the compromised node needs to be revoked and some or all of the compromised keys might need to be replaced. These processes, together with the policies and techniques needed to support them, are called key management. The facts that WSNs (1) are generally not tamper-resistant; (2) operate unattended; (3) communicate in an open medium; (4) have no fixed infrastructure and pre-configured topology; (5) have severe hardware and resource constraints, present unique challenges to key management. In this article, we explore techniques for meeting these challenges. What distinguishes our approach from a routine literature survey is that, instead of comparing various known schemes, we set out to identify the basic cryptographic principles, or building blocks that will allow practitioners to set up their own key management framework using these building blocks.

INTRODUCTION

A WSN key management scheme consists of three main components: (1) key establishment; (2)

key refreshment; (3) key revocation. Key establishment is about creating a *session key* between the parties that need to communicate securely with each other. Key refreshment prolongs the effective lifetime of a cryptographic key, whereas key revocation ensures that an evicted node is no longer to able to decipher the sensitive messages that are transmitted in the network. A thorough understanding of what role these components play and how they integrate with each other is crucial to the design of key management frameworks. Just as importantly, the design has to follow these WSN-specific guidelines:

Design Principle 1*Favor computation over communication*: In general, we do not mind doing a little bit more computation just to save a few transmissions, as communication is three orders of magnitude more expensive than computation.

Design Principle 2*Minimal public-key cryptography*: Public-key algorithms remain prohibitively expensive on sensor nodes both in terms of storage and energy. The use of public-key cryptography should be kept to a minimum, if necessary at all.

Design Principle 3*Resilience*: Severe hardware and energy constraint suggests that security should never be overdone – on the contrary, tolerance is generally preferred to overaggressive prevention. This reasoning leads us to design key management schemes that, instead of trying to be perfectly secure, aim to be resilient.

Our goal in this article is to identify and introduce, based on these guidelines and the state of the art in the literature, key management building blocks for WSNs.

An aspect of key management that is often overlooked in the WSN literature is the *formal verification* of cryptographic protocols, that is, the use of formal methods in mathematics to prove or disprove the correctness of these protocols. In protocol verification, the two most important properties to verify are *secrecy* and *authentication*. However, these problems are well-known to be *undecidable* (there is no way to tell whether the property is valid) if we assume the intruder can construct an infinite number of messages, or if there can be an unbounded number of parallel sessions (i.e., parallel executions of the same protocol). One approach to make the problem decidable is to limit the number of parallel sessions. Much of the work that uses this strategy is based on constraint solving. Our secondary goal in this article is to give a primer on protocol verification via constraint solving, in the hope that protocol verification will become an integrated step in the design of key management schemes for WSNs in the future.

The rest of the article is organized as follows. As preliminaries, we will first introduce the notation for specifying cryptographic protocols. We will then discuss protocol verification by using constraint solving. We will then introduce the building blocks in the three areas: key establishment, key refreshment and key revocation. All protocols mentioned in the course of discussion will be verified using constraint solving. Finally, we will give a brief conclusion.

NOTATION FOR PROTOCOL SPECIFICATION

The notation in Table 1 is used to specify cryptographic protocols for the rest of this article.

Symbols	Meaning
A, B, …	Usually represent node A, B, …
N_A, N_B, …	Usually represent a nonce (random number) generated by A, B, …
K_{AB}	Usually represents a key shared between A and B
E(K, M)	Encryption of message M using key K
MAC(K, M)	Message authentication code (MAC) of message M using key K
PRF(K, M)	Pseudorandom function with key K applied to plaintext M
||	Concatenation operator
K'	New key for replacing K during re-keying

Table 1. Notation for specifying cryptographic protocols

It is important to note that when both E(K, M) and MAC(K, M) appear in the same message, the encryption actually uses a sub-key generated from K, and the MAC uses another sub-key, also generated from K. For example, given a pseudorandom function PRF(·,·) and a master key K, the encryption sub-key can be derived as PRF(K, 1), whereas the MAC sub-key can be derived as PRF(K, 2). The reason for not using K directly is that some cipher operation modes like the popular CBC are susceptible to *birthday attacks*: if we use the same key to transform more than $O(2^{m/2})$ plaintexts, it becomes likely that two or more of these plaintexts might map to the same ciphertext, allowing data forgery to occur. We say $O(2^{m/2})$ is the *birthday threshold*. Using different sub-keys for encryption and for authentication allows us to process more plaintexts before reaching the birthday threshold. Also, unforeseen problems may arise if the same key is used for encryption and authentication.

PROTOCOL VERIFICATION

A number of formal methods can be used for protocol verification, depending on the restriction we impose on the attacker model. If we limit the number of parallel sessions, we can model a protocol using the *strand space model* , and use constraint solving to verify its security properties efficiently. The strand space model can be understood informally as a mapping of the notions on the first column of Table 2, to the notions on the second column of Table 2.

Protocol	Strand space model	Example
Role: What a principal does in the protocol	Strand: A sequence of events	Initiator, responder, server
Complete run: A complete iteration of the protocol	Bundle: A set of strands — legitimate or otherwise — hooked together where one strand sends a message and another receives that same message, that represents a full protocol exchange	1. Initiator → Attacker: … 2. Attacker → Responder: … 3. Responder → Attacker: … 4. Attacker → Initiator: …

Table 2. The strand space model

Basically, a protocol consists of *roles* that exchange messages with each other, and the messages

that ‘fly’ back and forth between the roles can be visualized as *strands*. A *bundle* is basically a bunch of interleaving strands. A *system scenario* is a hypothetical instantiation of the protocol between some specified principals with a specified outcome. For example, we can specify a system scenario where the principals include one initiator, one responder, one server; we can then define their roles, and specify the outcome as the attacker getting the session key — all of these are our constraints. If we can find a bundle that satisfies these constraints, then we can say the protocol does not satisfy the secrecy requirement. Note the fact that a bundle cannot be infinite means we cannot model infinite number of parallel sessions. In WSNs, we are mainly after these three security requirements:

- Secrecy: A session key must only be known to the communicating nodes.
- Authentication (implies integrity): A key establishment protocol must end with every party properly authenticating the other parties it is communicating with. In other words, it must be impossible for any intruder *M* to impersonate another node *A* whose keys (used in the key establishment protocol) *M* does not have.
- Replay resistance: The meaning of replay attack on a role *R* is the possibility of unauthenticated parties to cause *R* to run, i.e. for *R* to process replayed messages. If *R* happens to maintain the states of every run, then it would be maintaining the incorrect states.

The beauty of this approach is that it can easily be implemented using Prolog. One example is CoProVe (http://wwwes.cs.utwente.nl/coprove). All the protocols that are given in this article in standard notation have been verified using CoProVe.

KEY ESTABLISHMENT

We start with the first component of key management: key establishment. In precise terms, key establishment is a process or protocol whereby a shared secret key becomes available to two or more parties, for subsequent cryptographic use. There are two types of key establishment protocols:

1. *key transport*, where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s); and
2. *key agreement*, where two or more parties derive a shared secret as a function of information contributed by, or associated with, each of the parties, (ideally) such that no party can predetermine the resulting value.

A *key pre-distribution* protocol is a key agreement protocol whereby the resulting established keys are completely determined a priori by initial keying material. Key pre-distribution is essential to WSNs because (1) it minimizes the exchange of information, i.e., communication; (2) it does not require any key distribution center (KDC). However, as we shall see, key pre-distribution is not the only key establishment technique used in WSNs, because due to the resource constraints of sensor nodes, we can rarely pre-distribute enough keying material such that any pair of nodes

would be able to establish a session key. We will look at some key pre-distribution schemes later.

In WSNs, key establishment is required to support these basic communication modes: (1) global broadcast, or flooding; (2) local broadcast; (3) unicast. Hence, we will discuss the key establishment protocols in the context of supporting these communication modes. Note that for each mode, we can in theory either use symmetric-key cryptography or public-key cryptography, but we are honoring by restricting ourselves to using symmetric-key cryptography. The following discusses how key establishment can be done to support the three basic communication modes.

Global broadcast

In doing a global broadcast, a node (sender) intends to broadcast a message to all other nodes (receivers) in the network. The security objective is to ensure the integrity, authenticity and optionally the confidentiality of the messages from the sender to the receivers. The sender cannot share a key with all the receivers, because then any of the receivers can forge messages. The sender also cannot share a different key with each of the receivers, because this solution is not scalable. Instead, the standard solution for integrity and authentication is μTESLA (the "micro" version of the Timed, Efficient, Streaming, Loss-tolerant Authentication Protocol) .

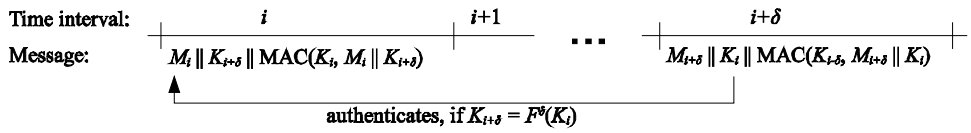

Figure 1. Keys are released according to a schedule in SPINS

To bootstrap the protocol, the sender first generates a *one-way key chain* (K_0, K_1, …, K_n), where $K_{i+1} = h(K_i)$, $i = 0,\ldots,n\text{-}1$ and $h(\ )$ is a *collision-resistant hash function* (i.e., a *one-way hash function* that is also collision-resistant), and distributes the root of the key chain K_n to the receivers securely. K_n is called the *commitment* of the key chain. For this protocol to work, the clocks of the sender and the receivers must be synchronized. The sender and the receivers divide time into intervals. If during time interval i, the sender broadcasts a message M_i, the sender appends M_i with a Message Authentication Code (MAC) of M_i generated with K_i. The receivers cannot authenticate M_i until δ intervals later, when the sender would broadcast K_i. (Figure 1). The receivers successfully authenticate the sender if $K_{i+j} = h^j(K_i)$, where K_{i+j} is the key released in time interval i-j (j can be any value between δ and n-i, assuming $n\text{-}i > \delta$). Note that the keys are released in the reverse order, because an attacker cannot re-generate the key chain in the reverse order due to the "one-wayness" of collision-resistant hash functions.

Since keys are distributed along with messages, μTESLA by itself cannot provide confidentiality. In this respect, a global key is usually used alongside μTESLA to provide data confidentiality.

Local broadcast

In doing a local broadcast, a node (sender) intends to broadcast a message to all its neighbors (receivers). The security objective is to ensure the integrity, authenticity and optionally the confidentiality of the messages from the sender to the receivers. As before, we may use μTESLA to provide integrity and authentication, and a *cluster key* (a key shared between a node and its neighbors) to provide confidentiality. Alternatively, we may relax the time synchronization requirement, because the receivers are just one hop away from the sender. The following protocol to be described is originally designed for *passive participation* — a data communication paradigm in which a node would suppress its own transmission if it overhears its neighbor(s) transmitting similar data. This alternative protocol is essentially μTESLA, used with a cluster key, but without a key disclosure schedule. In this protocol, the sender distributes, as in μTESLA, a commitment of its key chain, and additionally a cluster key to the receivers (which are also the sender's neighbors) . The rationale behind using this key combination is as follows:

- if only the key chain is used, the keys in the key chain would have to be broadcast in the clear, and in the absence of time interval differentiation, a cluster-outsider would be able to forge messages using these keys;
- if only the cluster key is used, authentication of the sender cannot be achieved;
- but if used together, the cluster key can be used to encrypt messages as well as hide the key chain keys from cluster-outsiders; and at the same time, the key chain keys can be used for authentication.

The disadvantage of this protocol is that a malicious insider is still able to forge messages to other receivers. Note that this protocol is not suitable for global broadcasts because a global broadcast travels more than one hop, and the lack of time intervals allows a malicious upstream receiver to forge messages to downstream receivers.

Unicast

Unicast is one-to-one communication. The security objective is to ensure the integrity, authenticity and optionally confidentiality of the messages exchanged between two communicating nodes. Denote the two nodes by A and B. We only deal with the case where A and B are neighbors, because when A and B are multiple hops away, we can usually secure one hop at a time. Our goal is to establish a session key between A and B, which in the WSN literature is called a *pairwise key*.

Random key pre-distribution The prevalent strategy for establishing pairwise keys is *random*

key pre-distribution (RKP) (aka *probabilistic key sharing*). The general idea is to prepare a pool of keying material, called the *key pool*; and to each sensor node, distribute a random fixed-size subset of keying material from the key pool. The keying material belonging to a node is called the node's *key ring*. Denote the key pool size as P and key ring size as K. Having potentially different subsets of the key pool, two neighboring nodes can only establish a pairwise key at a certain probability that is related to P and K; that is, a node may not be *securely* connected to all its neighbors. However by adjusting P and K, it is possible to make a network securely connected with high probability.

In RKP, this is how two nodes establish a session key: When a node is added to the network, the node initiates *shared-key discovery*, by broadcasting a list of identifiers that identify the keys it has. The neighbors reply with their lists of key identifiers. By comparing the lists, the new node and its neighbors discover what key(s) they share. Session keys are then derived from the shared key(s), for example, by applying a PRF on the XOR of the shared key(s). The disadvantage of this approach is that it allows an attacker to find out which keys a node is holding, giving room to the attacker to attack strategically. An alternative approach is, instead of picking keying material randomly for a node, to pick the keying material according to the result of a PRF. For example, the key index of node A's i-th key is given by PRF(A, i). Using this approach, a node can by just knowing the ID of its neighbor, determine the indexes of its neighbor's keys.

Different variants of RKP can be instantiated depending on what we use as 'keying material':

- Symmetric key : The simplest case is to use a single symmetric key as keying material. In this case, every node is imprinted with K keys chosen at random from a key pool of size P. When a node A is compromised, A's keys may be used to compromise other secure channels that do not involve A, since the keys might be stored in some other nodes outside A's communication range as well.
- Polynomial : In this case, the key pool consists of P symmetric t-degree bivariate polynomials over a finite field $_q$, i.e., a pool of polynomials of the form $f(x,y)=\sum_{i,j=0}^{t} a_{ij}x^i y^j$ with $a_{ij} = a_{ji}$; $a_{ij}, x, y \in$ $_q$; and q is a prime chosen to be much larger than the number of nodes as well as the desired key length. Denote this set of polynomials by $\{f_1(x,y),...,f_P(x,y)\}$. Every node A is then imprinted with K polynomial shares $f_{i_1}(A,y),...,f_{i_K}(A,y)$, by choosing different $i_1, i_2, ..., i_K$ randomly from $\{1,...,P\}$. By shared-key discovery, node A and B can find out which polynomials they have in common. If that polynomial is $f_1(x,y)$, then without loss of generality, A and B can establish a session key with each other by calculating the key as $f_1(A,B)$, and as $f_1(B,A)$ respectively. When a node A is compromised, A's polynomial shares $f_{i_1}(A,y),...,f_{i_K}(A,y)$ can only be used to compromise secure channels that involve A, unless the attacker manages to compromise t+1

shares of one of the shared polynomials.

- Matrix : In this case, the key pool consists of P matrices M_1, M_2, ..., M_P of size $N\times(t+1)$ over finite field $_q$, where N is the expected total number of nodes in the network; t is a security parameter; and q is a prime chosen to be much larger than N as well as the desired key length. The matrices are generated in three steps: First, a Vandermonde-like matrix G of size $(t+1)\times N$ over finite field $_q$ is generated using a primitive element s of $_q$:

$$G=\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ s & s^2 & s^3 & \dots & s^N \\ s^2 & (s^2)^2 & (s^3)^2 & \dots & (s^N)^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s^t & (s^2)^t & (s^3)^t & \dots & (s^N)^t \end{bmatrix} \quad (1)$$

At the second step, P random symmetric matrices D_1, D_2, ..., D_P of size $(t+1)\times(t+1)$ are generated. Thirdly and finally, the final matrices are calculated as $M_1 = (D_1\cdot G)^T$, $M_2 = (D_2\cdot G)^T$, ..., $M_P = (D_P\cdot G)^T$. G has the following useful properties: (1) since s is a primitive element and $N<q$, s, s^2, …, s^N are all unique and can be used as the nodes' IDs; (2) any $t+1$ columns of G are linearly independent. The following are what get distributed to the j-th node: (1) the j-th column of G, denoted $G(j)$; (2) the j-th row from each of $M_{i_1}, M_{i_2}, ..., M_{i_K}$, denoted $M_{i_1}(j), M_{i_2}(j), ..., M_{i_K}(j)$, where different i_1, i_2, ..., i_K are randomly chosen from $\{1,...,P\}$. Therefore in theory, each node has to store 1 matrix column and K matrix rows; but in practice, each node only has to store the 2nd element of its assigned column and K matrix rows, because all elements of the same column are just different powers of the 2nd element of the column. For example, the 1st node only has to store s, the 2nd node only has to store s^2 and so on. By shared-key discovery, node i and j can find out which matrices they have in common. If that matrix is M_1, then without loss of generality, i and j can establish a session key with each other by calculating the key as $M_1(i)G(j)$, and as $M_1(j)G(i)$ respectively. Note that $M_1G = G^TD_1{}^TG$ is symmetric, hence $M_1(i)G(j) = (M_1G)_{ij} = (M_1G)_{ji} = M_1(j)G(i)$, i.e., node i and node j are able to derive the same session key. When node j is compromised, node j's matrix rows $M_{i_1}(j), M_{i_2}(j), ..., M_{i_K}(j)$ can only be used to compromise secure channels that involve node j, unless the attacker manages to compromise $t+1$ rows of M_{i_1} or M_{i_2} or ... or M_{i_K}, because any $t+1$ rows of M_{i_1} or M_{i_2} or ... or M_{i_K} are linearly independent. This technique is actually inspired by *maximum distance separable* code.

We now consider the case where A and B do not share any key, but each has a secure link to a common neighbor S. In this case, A and B can still establish a session key through S acting as a

trusted third party. The following key transport protocol can be used to establish a session key K_{AB} between A and B via S:

1. $A \rightarrow S$: $N_A \parallel B \parallel \text{MAC}(K_{AS}, N_A \parallel B)$
2. $S \rightarrow A$: $\text{E}(K_{AS}, \text{E}(K_{BS}, N_S \parallel K_{AB})) \parallel \text{MAC}(K_{AS}, N_A \parallel B \parallel \text{E}(K_{BS}, N_S \parallel K_{AB}))$
3. $A \rightarrow B$: $A \parallel \text{E}(K_{BS}, N_S)$
4. $B \rightarrow S$: $B \parallel N_B \parallel A \parallel \text{MAC}(K_{BS}, N_S \parallel B \parallel N_B \parallel A)$
5. $S \rightarrow B$: $\text{E}(K_{BS}, K_{AB}) \parallel \text{MAC}(K_{BS}, N_B \parallel A \parallel \text{E}(K_{BS}, K_{AB}))$
6. $B \rightarrow A$: Ack, $\text{MAC}(K_{AB}, \text{Ack})$

This protocol has been verified with CoProVe to be (1) secure with respect to the secrecy of K_{AB}, (2) secure in the mutual authentication between A and B, and (3) secure against replay attacks on S .

LEAP+ An alternative scheme to RKP, as part of LEAP+ , is as follows:

1. First, embed an initial key K_{IN} in every node.
2. Upon bootstrapping, every node A derives its own master key as $K_A = \text{PRF}(K_{\text{IN}}, A)$, and set its timer to fire at time T_{min} later. T_{min} is the estimated minimum amount of time for an attacker to compromise a node. A sends out a HELLO message containing its ID.
3. As long as the timer has not fired, if A hears a HELLO message from a neighbor B, it will derive the pairwise key as $K_{BA} = \text{PRF}(\text{PRF}(K_{\text{IN}}, A), B)$. If B receives A's HELLO message first, then the pairwise key would be $K_{AB} = \text{PRF}(\text{PRF}(K_{\text{IN}}, B), A)$ instead.
4. When the timer fires, K_{IN} is erased from memory.

This scheme is however only useful for static networks, since after K_{IN} is erased, a node can no longer derive pairwise keys.

EBS Exclusion Basis Systems (EBS) is a variation of the symmetric-key version of RKP . Basically, instead of choosing K out of P keys at random, EBS chooses K out of P keys uniquely for each node, so there are only $P!/[K!(P-K)!]$ ways of choosing, and there can only be a maximum of $P!/[K!(P-K)!]$ nodes. By picking $K > P/2$, EBS makes sure every pair of nodes share at least one key, hence guaranteeing the network is connected. The drawback of this scheme is that, when a node is compromised, only P-K keys, or less than half of the keys in the key pool remain intact. Because of this, a WSN that uses EBS is most often compartmentalized into clusters, with a different key pool assigned to each cluster.

KEY REFRESHMENT

As mentioned, different sub-keys are used for encryption and for authentication because that would allow the birthday threshold to be reached more slowly, but the birthday threshold will eventually be reached. The standard solution to further delay the birthday threshold from being reached is key refreshment, i.e., the process of refreshing shared secrets periodically as a mean to

increase the birthday threshold (the cryptography literature generally uses 'key refreshment' and 're-keying' interchangeably, but we reserve 're-keying' for the process that follows key revocation). There are two mainstream approaches :

1. Parallel re-keying: We start with keys $K_{enc,0}$ and $K_{mac,0}$. The i-th (i = 1, 2, ...) refreshed keys are PRF($K_{enc,0}$, i) and PRF($K_{mac,0}$, i). Note: $K_{enc,0}$ and $K_{mac,0}$ can be generated from the same master key K_0 via PRF(K_0, 1) and PRF(K_0, 2).
2. Serial re-keying: We start with key K_0. The 1st refreshed keys are PRF(K_0, 1) and PRF(K_0, 2), respectively for encryption and MAC. The i-th (i = 2, 3, ...) refreshed keys are $\underbrace{\mathrm{PRF}(\mathrm{PRF}(\ldots\mathrm{PRF}}_{i-1 \text{ times}}(K_0,0)\underbrace{\ldots,0)}_{i\text{-}1 \text{ times}},1)$ and $\underbrace{\mathrm{PRF}(\mathrm{PRF}(\ldots\mathrm{PRF}}_{i-1 \text{ times}}(K_0,0)\underbrace{\ldots,0)}_{i\text{-}1 \text{ times}},2)$, again respectively for encryption and MAC.

The advantage of using these approaches is as follows. Suppose the key length is k. If the session key is not refreshed, the birthday threshold is $2^{k/2}$. If the session key is refreshed every $2^{k/3}$ function invocations (where 'function' is either encryption or MAC), the session key can be refreshed $2^{k/3}$ times before birthday attack is likely to succeed. In other words, the birthday threshold is increased from $2^{k/2}$ to $2^{2k/3}$.

For WSNs, serial re-keying is preferred, because in parallel re-keying, the counter i and the key K_0 have to be stored, and if a node is compromised, these information would allow an attacker to generate all past keys *in addition to* future keys. In other words, parallel re-keying does not provide *forward security*. On the other hand, in serial re-keying, only the term $\underbrace{\mathrm{PRF}(\ldots\mathrm{PRF}}_{i-1 \text{ times}}(K_0,0)\underbrace{\ldots,0)}_{i\text{-}1 \text{ times}}$ needs to be stored, and this does not allow any past key to be generated due to the non-invertibility of PRF.

KEY REVOCATION AND RE-KEYING

Key revocation is the process of removing keys from operational use prior to their originally scheduled expiry, for reasons such as node capture. When a node is found to be compromised, a key revocation list is constructed and broadcast using μTESLA to the whole network. The list contains the ID of the compromised node, and optionally the indexes of the node's keys — these keys are keys from the key pool, and there is a mechanism for calculating these indexes based on the node ID as described previously, so storing the key indexes is optional. The process of removing keys is usually accompanied by re-keying. Because of this, the main challenge for doing key revocation efficiently is to do re-keying efficiently. Let us consider the types of keys that need to be replaced in case of a key revocation:

1. Global broadcast keys: In the context of μTESLA, the key chain commitments that reside in the nodes do not need to be replaced, because all the nodes do is to wait for new keys from the key chain to be disclosed anyway. On the other hand, the global key needs to be replaced.
2. Local broadcast keys: Similarly, only the cluster key needs to be replaced.
3. Unicast keys: There are two scenarios: either the revoked keys are only used for the secure

channels that involve the evicted node(s), or the keys might actually be used elsewhere in the network for the secure channels that do not involve the evicted node(s) at all. The first scenario applies to LEAP+ , the polynomial-based and matrix-based RKP schemes, whereas the second scenario applies to EBS and the symmetric-key-based RKP scheme because a key in these schemes is potentially shared by nodes distributed all over the network (all these schemes are mentioned in the last section). For EBS, re-keying is essential, because every node contains more than half of the keys from the key pool. For symmetric-key-based RKP, re-keying is less urgent, because in this case, a node's key ring is typically much smaller than the key pool size. Therefore, as long as the compromised keys from the key pool ("pool keys" for short) are properly revoked, the network should only suffer from reduced connectivity (counting only secure links).

Now the types of keys to be replaced are known, the next issue to consider is how the new keys are generated and transported to the target nodes. At first sight, it seems that to renew the global key, there is a vast amount of literature on *secure group communication* that we can borrow techniques from. However, these techniques do not translate well to WSNs, mainly because they do not consider the multi-hop transportation of the new keys to the nodes. Furthermore, for WSNs, the logical first step is to renew the compromised pool keys, because the pool keys are used to derive pairwise keys, and the pairwise keys in turn are used to transport other keys. The following describes the procedures:

- The new pool keys can either be generated centrally or in a distributed fashion . In the latter case, some nodes are tasked with the generation of certain keys, e.g. the i-th node generates the i-th pool key. Either way, the problem is getting the new keys to the right nodes. As mentioned, re-keying is essential for EBS. When a node is compromised, P-K out of P keys in the key pool remain secure, and all uncompromised nodes must have at least one of these P-K keys (this is not the case for RKP!). Suppose without loss of generality the compromised keys are $K_1,..,K_m$. For each intact key K_i ($i = m+1,\ldots,P$), the message $E(K_i, E(K_1, K_1') \| \ldots \| E(K_m, K_m'))$ is generated, and μTESLA-broadcast to the network. Every node will then be able to replace their compromised keys and derive new pairwise keys. On the other hand, for symmetric-key-based RKP, re-keying is not crucial, and is actually not efficient to execute.
- The new cluster keys are generated by the nodes themselves. After setting up a pairwise key with a neighbor, a node transmits its cluster key to the neighbor, encrypted using the new pairwise key.
- The new global key is generated centrally and subsequently broadcast to the network. The generator does the following :
  1. The generator generates the new global key as K_g'.
  2. The generator broadcasts the hash of K_g', $h(K_g')$, to the network using μTESLA. Every node in the network caches $h(K_g')$. This hash will be used later to verify K_g'.
  3. The generator broadcasts K_g' to its neighbors encrypted with its cluster key. The neighbors individually verify K_g' with the hash $h(K_g')$ they have received earlier. The generator's neighbors then re-encrypt K_g' with their own cluster keys and broadcast the

re-encrypted $K_g^{'}$ to their respective neighbors. The process continues until $K_g^{'}$ reaches every node in the network. This flooding process can be made more efficient, by optimizing the underlying routing protocol, but the principle remains the same.

CONCLUSION AND FUTURE DIRECTIONS

In this article, we introduce the key management building blocks for WSNs based on a clear set of design guidelines and the state of the art in the literature. Along with our discussion, we stress the importance of protocol verification, giving one sample protocol that has been verified. Future key management architectures can be designed based on these building blocks. Integrating these components however is a challenging task, as there are many aspects to consider. For example, an energy-efficient key management architecture should be optimized for the underlying routing protocol and vice versa. Secure data aggregation also needs to be taken into account. Meanwhile, existing building blocks can be further improved. In fact, the polynomial and matrix technique in random key pre-distribution can be further generalized; and re-keying for symmetric-key-based random key pre-distribution is actually a difficult problem. Mostly importantly, the perpetual quest is to lower the resource requirements of key management.

ACKNOWLEDGEMENT

This work is sponsored by the ARC Research Network on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), and DEST International Science and Linkage Grant.

REFERENCES

[1] M. Abdalla and M. Bellare. Increasing the lifetime of a key: A comparitive analysis of the security of rekeying techniques. In T. Okamoto, editor, Advances in Cryptology – ASIACRYPT 2000, volume 1976 of LNCS, pages 546–565. Springer-Verlag, 2000.

[2] W. Du and J. Deng and Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili. A pairwise key predistribution scheme for wireless sensor networks. ACM Trans. Inf. Syst. Secur., 8(2):228--258, 2005.

[3] L. Eschenauer and V.D. Gligor. A key-management scheme for distributed sensor networks. In Proc. 9th ACM conference on Computer and communications security, pages 41–47. ACM Press, 2002. ISBN 1-58113-612-9.

[4] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why

is a security protocol correct? In Proceedings of The 1998 IEEE Symposium on Security and Privacy, pages 160-171. IEEE Computer Society Press, 1998.

[5] S.-R. Kim. Scalable hash chain traversal for mobile devices. In Computational Science and Its Applications ICCSA 2005, volume 3480 of LNCS, pp. 359-367. Springer-Verlag, 2005.

[6] Y.W. Law, R. Corin, S. Etalle, and P.H. Hartel. A formally verified decentralized key management architecture for wireless sensor networks. In the 4th IFIP TC6/WG6.8 International Conference on Personal Wireless Communications (PWC 2003), volume 2775 of LNCS, pages 27–39. Springer-Verlag, September 2003. ISBN 3-540-20123-8.

[7] D. Liu, P. Ning, and R. Li. Establishing pairwise keys in distributed sensor networks. ACM Trans. Inf. Syst. Secur., 8(1):41 77, 2005.

[8] M. Moharrum and M. Eltoweissy and R. Mukkamala. Dynamic combinatorial key management scheme for sensor networks. Wireless Communications and Mobile Computing, volume 6, issue 7, pp. 1017-1035. John Wiley & Sons, Ltd., 2006.

[9] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar. SPINS: Security Protocols for Sensor Networks. In Proceedings of the 7th Ann. Int. Conf. on Mobile Computing and Networking, pages 189–199. ACM Press, 2001. ISBN 1-58113-422-3.

[10] S. Zhu, S. Setia, and S. Jajodia. LEAP+: Efficient security mechanisms for large-scale distributed sensor networks. ACM Trans. Sen. Netw., 2(4):500-528, 2006.