KALwEN: A New Practical and Interoperable Key Management Scheme for Body Sensor Networks

Yee Wei Law*
The University of Melbourne

Giorgi Moniava†
Zheng Gong†
Pieter Hartel†
University of Twente

Marimuthu Palaniswami*
The University of Melbourne

ABSTRACT

Key management is the pillar of a security architecture. Body sensor networks (BSNs) pose several challenges – some inherited from wireless sensor networks (WSNs), some unique to themselves – that require a new key management scheme to be tailor-made. The challenge is taken on, and the result is KALwEN, a new lightweight scheme that combines the best-suited cryptographic techniques in a seamless framework. KALwEN is user-friendly in the sense that it requires no expert knowledge of a user, and instead only requires a user to follow a simple set of instructions when bootstrapping or extending a network. One of KALwEN's key features is that it allows sensor devices from different manufacturers, which expectedly do not have any pre-shared secret, to establish secure communications with each other. KALwEN is decentralized, such that it does not rely on the availability of a local processing unit (LPU). KALwEN supports global broadcast, local broadcast and neighbor-to-neighbor unicast, while preserving past key secrecy and future key secrecy. The fact that the cryptographic protocols of KALwEN have been formally verified also makes a convincing case.

1. INTRODUCTION

Specialized WSNs called BSNs are facilitating a revolution in the healthcare industry. A BSN is a wireless network of small, low-cost, biosensors worn by a human user, for the purpose of monitoring the user's physiological parameters, e.g., ECG, EMG, EEG, SpO_2, blood pressure etc.

The classic architecture of a BSN consists of a network of biosensors and an on-body *local processing unit* (LPU), usually in the form of a PDA or mobile phone, directly or indirectly connected to a remote server storing the user's records [26]. However, for practical purposes that include avoiding a single point of failure, in our reference architecture, we do not stipulate the presence of an LPU. The number of nodes is up to a few dozens. All nodes are capable of far-field (radio) communication, some nodes in addition maybe wired or capable of near-field communication – device-to-device or intra-body (the latter is also called *body-coupled communication* [40]).

The security and privacy problems related to healthcare systems are real [2]. As a recent study has demonstrated, medical devices that do not support any confidentiality and authentication function are prone to eavesdropping and attacks [18]. Solving these problems requires data confidentiality and authentication. Providing data confidentiality and authentication in turn requires a key management scheme to put the cryptographic keys in place. A practical key management scheme has to take into account the following constraints of a BSN [1]:

(i) Usability: The most basic functional requirement is that the key management process has to be autonomous enough such that barring some simple, fool-proof procedures a user has to follow, it requires no expert knowledge whatsoever on the part of both the user and the medical personnel.

(ii) Interoperability: Sensor devices from a manufacturer should interoprate with devices from other manufacturers. For example, their IDs should be globally unique. Also, due to their different origins, these nodes should not be required to store any pre-shared secret. The nodes must be able to establish session keys without relying on specific sensing capabilities (more in Section 2).

(iii) Hardware: BSNs experience the same hardware constraints as WSNs do, i.e., limited computational power, limited memory, limited bandwidth and limited energy. Public-key algorithms should thus be avoided as much as possible. Moreover, a sensor node is typically not tamper-resistant.

(iv) LPU: When BSNs evolve from WSNs, they lose the dependability of a base station, because in BSNs, the equivalent of a base station – an LPU – might not exist. After all,

*The first and last authors are supported by the Australian Research Council Research Network on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), and the DEST International Science and Linkage Grant.

†The authors acknowledge the financial support of SenterNovem for the ALwEN project, grant PNE07007; and thank Hermie Hermens for his input on the use case.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

[1]While it is paramount for implantable medical devices to provide emergency accessibility mechanisms, BSNs are generally only for monitoring and it is likely that an emergency personnel has and should prefer to use their own equipments to diagnose the user, so emergency accessibility of a BSN is less critical.

an LPU is potentially costly, is cumbersome to carry around and may present itself as a single point of vulnerability. The hardware constraint suggests that we should not expect any expert to manage cryptographic keys, whereas the LPU constraint suggests even if there is such an expert, there might not be an LPU with the proper user interface for him/her to manage keys.

(v) Multihop: Although the human body is relatively small-scale compared to the radio range of a typical sensor node, due to the unpredictable nature of radio communications, we must assume some nodes are not directly connected, i.e., some nodes need to communicate across multiple hops.

Our objective is to propose a key management scheme under the above constraints. Our contribution is KALwEN (short for Key management for Amblient Living with Embedded Networks). KALwEN combines the most relevant techniques in the literature in a complete framework, and is possibly the first proposal of such nature. KALwEN satisfies the above-mentioned requirements/ constraints, and supports the three basic communication modes: global broadcast, local broadcast and neighbor-to-neighbor unicast, while preserving past key secrecy and future key secrecy. Using formal verification, we show that the cryptographic protocols of KALwEN are secure.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the building blocks of KALwEN. Section 4 gives an overview of KALwEN. Section 5 describes the threat model and the assumptions against which KALwEN is built. Sections 6 to 10 specify the protocols for the factory phase, bootstrap phase, deployment phase and operation phase of a BSN. Section 11 presents our formal verification results. Finally, Section 12 concludes.

2. RELATED WORK

BSNs, being descended from WSNs, can benefit from the many ideas that have already been proposed for WSNs. Perrig et al. [33] pioneer the use of one-way hash chains in the form of μTESLA to achieve authenticated broadcasts in WSNs. Anderson et al. [3] suggest during the bootstrapping phase of a network, keys can be exchanged in the clear. LEAP+ [45] and its improved variants [14] suggest that during the short time before a node is likely to be compromised, a node can use its so-called initial key (a system-wide transitory master key) to establish secure channels with all its neighbors, before deleting the initial key. Kuo et al. [21] propose establishing a pairwise key between a base station and each node in a Faraday cage. For bootstrapping multiple nodes at the same time, they recommend using software attestation [38] to make sure all the nodes are uncompromised before the pairwise keys are transmitted in the clear within the Faraday cage. In combination with Diffie-Hellman key exchange, software attestation can be used for key establishment [37]. The catch is, software attestation works only (i) when the verifier has the same software as the prover's, and (ii) when a verifier knows the exact make and model of a sensor node's CPU.

The random key pre-distribution paradigm is pioneered by Eschenauer et al. [17] and later extended by Liu et al. [25] and Du et al. [16]. The idea is to prepare a pool of 'keying material' (which can be single symmetric keys [17], polynomials [25] or matrices [16]), called the *key pool*; and to each sensor node, distribute a fixed-size subset of keying material randomly chosen from the key pool (a node's keying material is called the node's *key ring*). Two neighbors are able to establish a pairwise key when they share at least a key. Two neighbors that do not share a key must use a common trusted neighbor to establish a pairwise key – the number of neighbors that are involved in the process is called the *key-path length*. The paradigm is particularly useful for limiting key storage per node in large-scale WSNs.

Later extensions of random key pre-distribution introduce deterministic, combinatorial designs – each key ring is drawn from the key pool in a deterministic fashion according to some combinatorial rules (for brevity, we call this paradigm combinatorial key pre-distribution). Various combinatorial designs have been proposed [23, 11, 9]. With respect to a fixed key ring size, the various designs enable different trade-offs on the following parameters: probability of key-share, average key-path length, resilience to node capture. The conclusion so far is that except for really small networks (networks that consist of at most n nodes and where a sensor node can store at most $n-1$ pairwise keys), combinatorial key pre-distribution seems to be the direction forward for WSNs, and by extension BSNs.

Ideas from ubiquitous computing can also be useful. The literature of this area primarily focuses on *secure pairing* [20]. Secure pairing is a solution to the key establishment problem between two strangers with no prior shared secret. The standard solution is to use a band-limited side channel to exchange short secrets and based on the short secrets establish a session key. The usage of four-digit PINs in Bluetooth is a classic example, although a relatively insecure one. Another example is to exchange keys in the open, e.g., by signaling in the *source* field of the packets, as long as an attacker cannot tell where the packets originate from [1, 8]. Another paradigm is to derive a common key by sampling similar side channels. For example, co-located devices experience similar radio environments [43]; devices shaken together can establish a common key [29]. Other examples that are based on this paradigm involve a pair of human users exchanging visual information, but these are not as useful [30, 7] for BSNs. By extension, two devices sensing the same or similar physiological signals should be able to derive a common key. This biometric extension is first proposed by Cherukuri et al. [12] for BSNs. Later work investigates the feasibility of using the heartrate variability [4], the interval pulse [34], the electrocardiogram [44] as the biometrics. When all the nodes in a network are capable of sensing the same type of signal, these results are applicable, and all the nodes can establish a common group key. Our goal is to handle the general case, i.e., where there is a chance that there is no overlap in sensing capabilities among the nodes.

Some architectures rely on a local base station to authenticate sensor nodes biometrically [27, 28]. These architectures are more useful for dedicated healthcare facilities than for ambient-assisted living scenarios.

3. PRELIMINARIES

This section introduces the 'building blocks' that we use to construct KALwEN. Table 1 partially summarizes the symbols used in this paper.

(a) Elliptic curve Diffie-Hellman (ECDH) Without any prior shared secret, two nodes can establish a session key using the Diffie-Hellman (DH) key agreement protocol [15].

Table 1: Partial list of symbols

$\{\cdot\}_K$	Encryption function using key K
$[\cdot]_K$	Message authentication function using key K
$H(\cdot)$	Cyptographic hash function
$\|$	Concatenation operator
$\xleftarrow{R}$	Randomly and uniformly chosen from
$\ggg$	Right shift operator
V_{SFC}	Set of all nodes excluding the key distribution center s in a Smart Faraday Cage
$\mathcal{N}_v$	Set of all neighbors of node v
ID_v	ID of node v
NID	Network ID
$(q,\ FR,\ a,\ b,\ G,\ n,\ h)$	Domain parameters (Section 3)
$\mathcal{K}$	Key space
K_M	Membership key (Section 8)
$\mathcal{P}$, P	Key pool (Section 2), $P = \|\mathcal{P}\|$
κ_v, K	Key ring of node v (Section 2), $K = \|\kappa_v\|$
$CD(\mathcal{P}, ID_v)$	A function that returns to the node v, its key ring and corresponding key indexes drawn from the key pool $\mathcal{P}$
KID	An array of key indices
K_G	Global key (Section 4)
K_v	Cluster key of v (Section 4)
K'	Renewed version of key K
N_v	Nonce sent by v
τ	Threshold of a secret sharing scheme
l	Number of keys in a one-way hash chain
m	Length of a hash
c	Length of a counter
q	Order of a finite field, or number of oracle queries, depending on the context

Over the years, the original DH protocol has been heavily extended. Among the numerous variants that exist in the literature, the variant discussed here is the elliptic curve Diffie-Hellman (ECDH) scheme [10, Section 6.1] using the elliptic curve cofactor Diffie-Hellman primitive (which is resistant gainst small subgroup attacks compared to the original primitive) [10, Section 3.3.2]. The security of ECDH hinges on the intractability of finding l such that $lG = Q$ given G, a point on an elliptic curve of large prime order, and Q, a scalar multiple of G. *Recent implementation results show that while ECDH is expensive, it is achievable at a time cost of the order of seconds, a ROM cost of under 20 KB and a RAM cost of around 2 KB, i.e., within the capabilities of a typical sensor node* [24].

An elliptic curve cryptosystem is built on a set of *domain parameters*. The standard notation for describing domain parameters is (q, FR, a, b, G, n, h), where q is the order of the finite field the elliptic curve takes its values from, FR is the field representation, a and b are the coefficients of the elliptic curve, G is the base point on the elliptic curve, and nh (n being a large prime) is the number of rational points on the elliptic curve.

Suppose node u has private/public key pair d_u/d_uG whereas node v has private/public key pair d_v/d_vG. Then the session key K_{uv} between u and v is derived as follows: (i) u and v exchange their public keys, (ii) u (v) computes $(x, y) = hd_ud_vG$ ($= hd_vd_uG$), (iii) if $(x, y) = (0, 0)$ then stop, otherwise $K_{uv} = KDF(x)$, where $KDF()$ is a key derivation function (that typically invokes a hash function multiple times). The reason for using $KDF()$ is that x may have some bits that can be predicted with non-negligible advantage [22].

(b) Combinatorial key pre-distribution We are solely interested in the type of scheme that ensures that any pair of nodes have at least one key in common. While this type of schemes are in general less resilient to node capture attacks, it is important from the users' perspective that network deployments are snappy, and furthermore, an attacker cannot capture too many nodes without alarming the user. High probability of key-share therefore takes precedence over resilience. Suppose the key ring size is fixed at K. The simplistic approach of setting the key pool size $P = 2K - 1$ and assigning one of the $\binom{2K-1}{K}$ combinatorial patterns to a node would make sure any pair of nodes share *at least* a key [31], but the resilience of this scheme is unsatisfactory (capturing a node compromises half of the key pool). A better approach is *symmetric designs* [9]. For a key ring size of K, there exists a symmetric design that supports up to $(K-1)^2 + (K-1) + 1$ nodes, and that ensures any pair of nodes share exactly one key. The supported network size seems to be sufficient for BSNs, for example, when $K = 10$, the maximum network size is 91; when $K = 20$, the maximum network size is 381, which is more than the size expected of a BSN. In case the maximum supported network size is exceeded, a symmetric design can be extended to support up to $\binom{(K-1)^2+(K-1)+1}{K}$ nodes [9], at the cost of reducing the probability of key-share. We write $CD()$ to denote a function that outputs the key ring and key identifiers allocated from a key pool to a node, i.e., $(\kappa_v, KID_v) = CD(\mathcal{P}, ID_v)$. Each 'key' in a key ring in this context can actually be a single symmetric key or a polynomial [36], depending on the desired level of trade-off: using polynomials gives better resilience at the cost of memory usage. The decision on whether to use polynomials or symmetric keys can be safely deferred to the time of actual implementation without sacrificing the soundness of KALwEN.

(c) One-way hash chain μTESLA [33] is the de facto *symmetric-key* solution for authenticated broadcast. To bootstrap the protocol, the sender s first generates a one-way hash chain $\{H_{s,0}, H_{s,1}, ..., H_{s,l-1}\}$, where $H_{s,i} = H(H_{s,i+1})$ $(i = 0, ..., l-2)$, $H()$ is a cryptographic hash function (the security requirements of which will be determined shortly) ; and distributes $H_{s,0}$ (called the commitment of the hash chain) to the receivers securely. For this protocol to work, the sender and the receivers must synchronize their clocks, at least loosely. The sender and the receivers divide time into intervals. Whenever the sender broadcasts a message M_i during time interval i, the sender always appends $[M_i]_{H_{s,i}}$ to M_i. The receivers cannot authenticate M_i until δ intervals later, when the sender would broadcast $H_{s,i}$ in the clear. The receivers successfully authenticate M_i if (i) there exists a past key $H_{s,j} = H^{i-j}(H_{s,i})$ $(0 \le j < i)$; and (ii) $H_{s,i}$ generates $[M_i]_{H_{s,i}}$.

Bradford et al. [5] propose using k-wise independent hash chain functions for μTESLA, but they also discover the probability of choosing such a function from the set of all functions from $\{0,1\}^m$ to $\{0,1\}^m$ is impractically small when k is large. So in practice, μTESLA should not be used in its original form. Instead, a larger domain should be used; for example, by using a separate chain of salts [32], i.e., $H_{s,i} = H(\text{salt}_{s,i+1} \| H_{s,i+1})$. Meanwhile, Bradford et al. [6] propose using a separate chain of counters which do not need to be transmitted; they also propose $H_{s,i} = H(H_{s,i+1} \| H_{s,i+2} \| ...)$. Throughout our ensuing discussion, for clarity, we continue

to write μTESLA in its original form, with the implicit understanding that in execution, μTESLA should be implemented using one of the aforementioned more secure proposals, and with an *always preimage-resistant* (aPre-secure [35]) hash function that is indifferentiable from a random oracle [13].

(d) Threshold secret sharing The standard technique for sharing a secret between n parties such that any τ or more parties can reconstruct the secret is (τ, n)-threshold secret sharing [39]. If the secret to be shared is S, then a random polynomial over a finite field of order q, $f(x) \in \mathbb{Z}_q[x]$, is generated, such that $f(x) = \sum_{i=0}^{\tau-1} a_i x^i \in \mathbb{Z}_q[x]$, where $a_0 = S$, and q is larger than the largest possible value of S. The shares are distributed as $f(ID_i)$ $(i = 1, ..., n)$, where ID_i's are the individual IDs of the participants. Any τ or more shares are enough to reconstruct S via Lagrange interpolation. $\tau - 1$ or less shares reveal no information at all about S.

4. SCHEME OVERVIEW

We begin the overview by identifying the essential keys that need to be established. Then we describe the architecture of a node in terms of its life cycle, hardware architecture and finite state machine. At the end of this section, we also describe the equipments that are needed to support the nodes.

KALwEN's mission is to support authentication in these communication nodes: (i) neighbor-to-neighbor unicast, (ii) local broadcast, (iii) global broadcast. The first two communication modes are the basic operations of a medium access protocol. The latter is the basic operation of a routing protocol.

To support confidential and authenticated neighbor-to-neighbor unicast, every pair of neighbors need a *pairwise key* [45].

To achieve confidential and authenticated local broadcast, a node needs to share a *cluster key* and uses a *cluster hash chain* with its neighbors [45].

Confidential and authenticated global broadcast by a node requires the node to share a *global key* and use a *global hash chain* with all the nodes in the network [45].

Figure 1(a) illustrates (i) the global key and global hash chain used by the KDC s; (ii) the pairwise keys used by v and v's neighbors; and (iii) the cluster key and cluster hash chain used by v.

Node architecture We divide the life cycle of a node into the following phases:

- Factory phase: Happens after the node is manufactured and before the node is bootstrapped for the first time.
- Bootstrap phase: Happens when the user bootstraps the node in a controlled enviroment, and before the user deploys the node.
- Deployment phase: Happens after the user bootstraps the node, but before the node starts operating. This is the time when the node tries to discover its neighbors.
- Operation phase: Happens after the node has discovered all its neighbors.
- Limbo phase: Happens after the node is removed from the network. Before the node can be deployed again, it must be re-bootstrapped.

When a node is first switched on, its bootloader copies the operating system (OS) from the external program memory to the internal program memory, and then transfers control to the OS in the internal program memory. Whenever a node is reset, the OS in its internal program memory is erased, and its bootloader copies the OS from the external program memory to the internal program memory afresh. All ephemeral cryptographic keys are stored in the RAM, so that when a node is switched off or reset, all keys are lost.

Figure 1(b) depicts the simplified finite state machine (FSM) of a node. The innerworking of this FSM shall become clear as we describe the protocols in later sections.

Equipments To bootstrap a network, and to add a node to a network, two equipments are needed: a 'Smart Faraday Cage' (SFC) and an RF jammer. A Faraday cage is a metal enclosure for shielding equipments within the cage against electromagnetic fields outside the cage. A Faraday cage, although not yet commercially available for BSNs, can already be purchased for individual mobile devices[2]. The kind of SFC we require is a Faraday cage with imbued intelligence such that it can also act as a KDC for the nodes to be bootstrapped, i.e., *the SFC and the KDC are the same entity.* The conceptual design of the SFC consists of a knob that can be set to one of the three modes: (i) 'Bootstrap' for bootstrapping nodes to form a new network; (ii) 'Standby' for writing everything in its volatile memory to its nonvolatile memory and going to sleep; (iii) 'Add' for bootstrapping nodes to be added to the previously bootstrapped network. The SFC has three indicators: 'Working', 'Done' and 'Error'. One security requirement of the SFC is that it must be dimensioned such that any two nodes in it must be within range of each other. The functional requirements of the SFC are defined by the protocols the KDC has to follow, as we describe below. A jammer can come in the form of a transceiver with minimum intelligence, that constantly transmits a noise signal. *It is to be emphasized that an SFC and a jammer are only needed when bootstrapping a network or adding a node to a network.*

5. THREAT MODEL AND ASSUMPTIONS

An attacker is computationally bounded. This is a standard cryptographic assumption, implying even if the attacker has access to supercomputers, its computing power is at most polynomial.

Communication-wise, there exists a range around a BSN outside which no attacker can eavesdrop on the messages of the BSN, even with advanced skills and equipments [19]. When an attacker is in range, the attacker can forge, intercept and arbitrarily manipulate any message, as prescribed by the standard model [41]. When an attacker is out of range, the attacker can only forge messages.

Hardware-wise, an SFC is tamper-resistant whereas a normal node is not. The SFC acts as a 'control center' for BSNs much like a base station acts as a center of command for WSNs. If the center of command is compromised, enforcing any form of security is meaningless. The essential rationale behind this assumption is that it is easier to safeguard the

[2]For example, `http://www.mobilecloak.com`

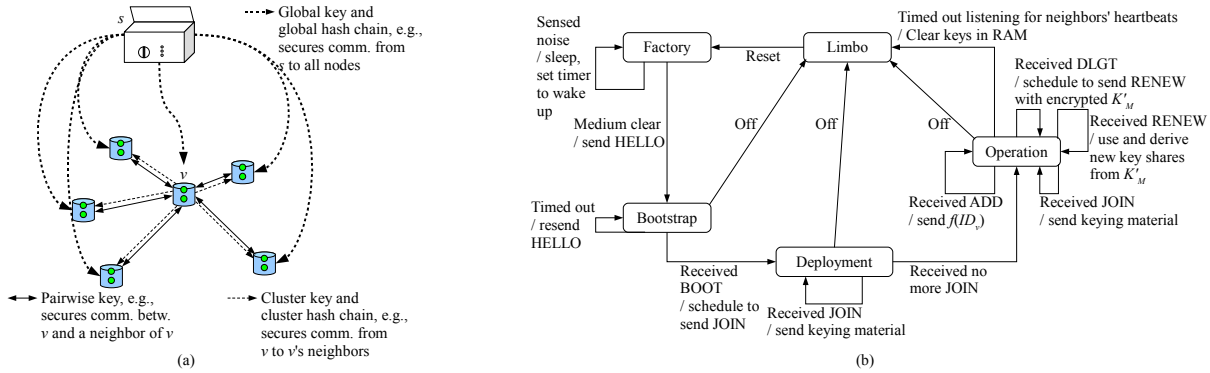

Figure 1: (a) Essential key types; (b) simplified finite state machine of a node

security of a single component rather than a host of smaller components that are far less physically manageable.

Physically, an attacker can get in-range with a BSN for an indefinite amount of time, but an attacker cannot try to remove τ or more nodes without the user noticing.

6. FACTORY PHASE

In the factory phase, every node v is embedded with the same set of domain parameters (q, FR, a, b, G, n, h) (Section 3).

For all the protocols of KALwEN, it is vital that a node can generate pseudorandom numbers with sufficient randomness. There are ways to collect entropy for this purpose, for example, by sampling the radio or on-board sensors. As a supplement, every node can be embedded with a unique seed, to be used as an input to the builtin pseudorandom number generators.

All of the above also applies to SFCs.

7. BOOTSTRAP PHASE

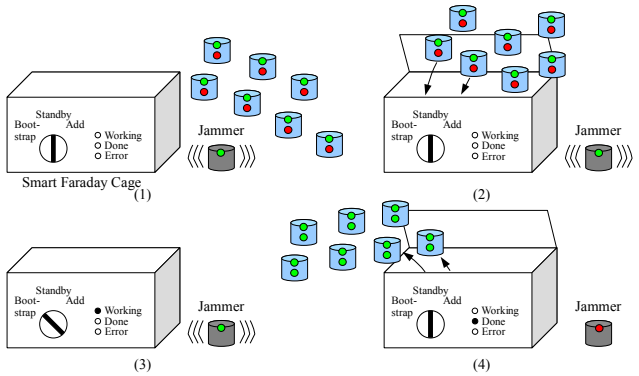

Figure 2: Procedure for bootstrapping nodes

The bootstrap process takes place in a controlled environment. Apart from the nodes themselves, the user needs two equipments: a 'Smart Faraday Cage' (SFC) and an RF jammer, which have already been described in Section 4.

The instructions a user has to follow are: (Figure 2): (1) switch on the jammer, then switch on and reset the nodes in close proximity of the jammer; (2) put the nodes in the SFC; (3) set the mode of the SFC to 'Bootstrap' and seal the SFC; (4) wait for the 'Done' indicator and then deploy the nodes. We describe the technical detail behind the instructions below.

Denote a node by v. When v is switched on, it senses the medium for a signal. If a noise signal is sensed, v sets a wake-up timer and go to sleep until the wake-up timer times out. This process continues until v wakes up and senses no noise signal. This is the time when v has been put into the SFC. Once v senses the channel is clear, it broadcasts a HELLO packet. All nodes contend for the medium to send their HELLO packets. Once the KDC stops hearing HELLO packets, based on (i) the number of distinct HELLO packets the KDC has received so far (which is $\mid V_{\text{SFC}} \mid$ if all nodes have sent their HELLO packets, but in anticipation for expansion, the KDC should add some margin to the network size), and (ii) the minimum size of a key ring K, the KDC computes the necessary key pool size and generates a key pool $\mathcal{P}$ of that size. With each node v, the KDC establishes a pairwise key K_{sv} using ECDH. For v, the KDC allocate κ_v, a block of K keys from $\mathcal{P}$. To v, the KDC then dispatches κ_v and other keying material encrypted using K_{sv}. Denote the KDC by s, and the protocol is as follows:

Protocol 1. (Closed environment in the SFC)

$$
\begin{aligned}
&\forall v \in V_{\text{SFC}}, \\
&\quad v \quad\quad : r_v \xleftarrow{R} [1, n-1] \\
&\quad v \rightarrow * \; : ID_v \| ID_* \| \mathsf{HELLO} \| r_v G \| N_v \\
&\forall v \in V_{\text{SFC}}, \\
&\quad s \quad\quad : r_s \xleftarrow{R} [1, n-1] \\
&\qquad\quad\;\; K_{sv} \text{ is derived per Section 3(a)} \\
&\qquad\quad\;\; K_M \xleftarrow{R} \mathcal{K} \text{ (detail later)} \\
&\qquad\quad\;\; (\kappa_v, KID_v) = CD(\mathcal{P}, v) \\
&\qquad\quad\;\; K_G \xleftarrow{R} \mathcal{K}, H_{s,l-1} \xleftarrow{R} \{0,1\}^m \\
&\qquad\quad\;\; H_{s,i-1} = H(H_{s,i}) \forall i \in \{1, ..., l-1\} \\
&\qquad\quad\;\; \theta = ID_s \| ID_v \| \mathsf{BOOT} \| r_s G \| N_s \| NID \\
&\qquad\quad\;\; \| \{K_M \| \kappa_v \| KID_v \| K_G\}_{K_{sv}} \| H_{s,0} \\
&\quad s \rightarrow v \; : \theta \| [N_v \| \theta]_{K_{sv}} \\
&\quad v \rightarrow s \; : [N_s]_{K_{sv}}
\end{aligned}
$$

Some notes about protocol listings:

- In Protocol 1, every message is explicitly prepended with a source field and a destination field. *Hereafter however, the source field and the destination field will be made implicit.* If a message is appended with a message authentication code (such as the message $s \rightarrow v$), the code is implicitly calculated over the source field and the destination field as well.

- Whenever we use the same key for encryption and message authentication in the same message, we are in fact using separate sub-keys derived from the same key. For example, for the last message in Protocol 1, we are actually using $[1]_{K_{sv}}$ and $[2]_{K_{sv}}$ as the encryption key and the message authentication key respectively, consistent with standard approach [33]. For brevity, we use K_{sv} to denote both sub-keys.

- While using message authentication code (MAC) is the standard technique for message authentication, an efficiency-enhancing alternative is to use $H(\text{key}\|\text{message})$, although at the expense of security.

As Protocol 1 does not authenticate public keys, it is open to impersonation attacks. However, if say a malicious node tries to impersonate v or the KDC, such action is immediately detectable by v or the KDC, because every node is within range of each other. If the attack is against v, v would immediately alert the KDC, and the KDC would immediately signal 'Error' to the user. If the attack is against the KDC, the KDC would directly signal 'Error' to the user. *Consequently, impersonation attacks and hence man-in-the-middle attacks are detectable in the specific environment of the SFC.*

K_M is called the *membership key* because it will be used to establish pairwise keys between neighboring nodes in the deployment phase. After broadcasting K_M, the KDC only keeps $H(K_M)$ instead of K_M itself. Doing so serves two purposes: (i) even when the KDC itself is compromised, an attacker cannot obtain K_M; (ii) the hash can be used in Protocol 3 to verify if the shares are correctly recovered (more on this in Section 9).

If the protocol goes well without any impersonation attack, after the KDC has finished bootstrapping all nodes, the KDC signals 'Done' to the user and the nodes are ready for deployment.

8. DEPLOYMENT PHASE

Neighbor discovery takes place during the deployment phase. Every node sets up a pairwise key with each of its neighbors. Using the pairwise keys, the node distributes its (i) cluster key and (ii) a commitment of its cluster hash chain to all its neighbors, per Protocol 2.

Protocol 2.

$\forall v \in V$,

$v \rightarrow *$: $\mathsf{JOIN}\|\{NID\|KID_v\}_{K_M}$

$\forall u \in \mathcal{N}_v$,

u : $K_{uv} \leftarrow H(x)$ where $x \in \kappa_u \cap \kappa_v$

$K_u \xleftarrow{R} \mathcal{K}, H_{u,l-1} \xleftarrow{R} \{0,1\}^m$

$H_{u,i-1} = H(H_{u,i}) \forall i \in \{1, ..., l-1\}$

$\theta = \{K_u\}_{K_{uv}}\|H_{u,0}\|N_u$

$u \rightarrow v$: $\theta\|[\theta]_{K_{uv}}$

v : $K_{uv} \leftarrow H(x)$ where $x \in \kappa_u \cap \kappa_v$

$v \rightarrow u$: $[N_u]_{K_{uv}}$

All legitimate nodes are supposed to have obtained K_M in the bootstrap phase. Any outsider without K_M is unable to join the network.

After neighbor discovery (signalled by a certain time-out), every node generates *deterministically* a function

$$f(x) = \sum_{i=0}^{\tau-1} a_i x^i \in \mathbb{Z}_q[x], \quad (1)$$

where $a_0 = K_M$, $a_i = H^i(a_0)$, and q is larger than both the largest possible membership key and the largest possible ID. Since $f(x)$ is generated deterministically, all nodes get the same coefficients a_i $(i = 0, ..., \tau - 1)$. Each node v then evaluates $f(ID_v)$, and discards all the coefficients, as well as K_M. This secret sharing scheme ensures that at least τ shares of K_M are required to reconstruct K_M and this is the principle behind the process of node addition, to be discussed later. Deleting K_M ensures that K_M cannot be used to perform illegitimate node addition.

At the end of the bootstrap phase, each node is capable of secure neighbor-to-neigbor unicast and secure local broadcast. Each node can also receive secure broadcast from the KDC, when the broadcast is relayed by a chosen node, as we shall discuss in the next section.

9. NODE ADDITION

Since a new node is *considered* trusted to be added to a network, there seems to be no incentive in enforcing *past key secrecy* (not to be confused with "perfect forward secrecy"), which is the requirement that a new member must not know old group (global) keys [41]. However, the new node may actually turn out to be rogue, in which case it is prudent to refresh the global key before admitting the new node into the network.

The easiest but most inefficient way to add a node to the network is to reset and bootstrap all nodes afresh. An alternative solution is to have new nodes pre-bootstrapped. For example, during the previous bootstrap phase, 20 nodes are bootstrapped but only 10 are deployed. The other 10 nodes that are not deployed are considered pre-bootstrapped. However, this solution stresses too much on foresight; and would not work at all if there are no extra nodes to start with in the first place.

For a new node to join the network, the user must undertake a procedure that is hard for an attacker but easy for him/herself to accomplish. Upon successful completion of the procedure, there should be a viable cryptographic means for the new node to establish the necessary keys for

supporting all the basic communication modes. Our rationale is to require the user to use a fair number of nodes to help bootstrap the new node, as under our assumption it is hard for an attacker to acquire that many nodes from the user without raising the user's suspicion.

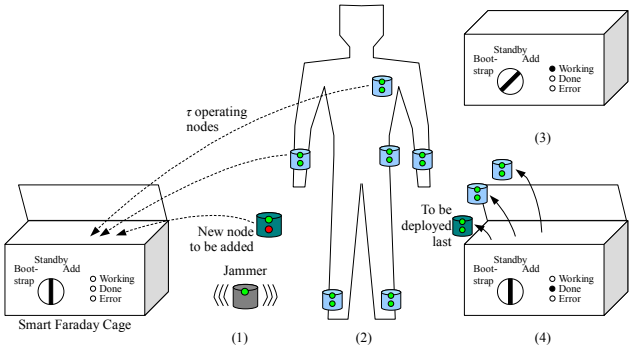

Figure 3: Procedure for adding a node

To add a node to the network, a user has to follow these instructions (Figure 3): (1) switch on and reset the new node in close proximity of the jammer, and put the new node into the SFC; (2) pick any τ operating nodes and put them into the SFC; (3) seal the SFC and set it to 'Add' mode as soon as the previous step is completed; (4) wait for the 'Done' indicator and then deploy the nodes, with the old nodes first and the new node last. We describe the technical detail behind the instructions below.

Denote the new node by w. When w is switched on, it senses the medium for a signal. If a noise signal is sensed, w sets a wake-up timer and goes to sleep until the wake-up timer times out. This process continues until w wakes up and senses no noise signal. This is the time when w has been put into the SFC. Once w senses the channel is clear, it broadcasts a HELLO packet. If there are more than one new node, the other new nodes will also contend for the medium and broadcast their HELLO packets. Once the KDC stops hearing HELLO packets, it broadcasts an ADD packet (reminder: the KDC has been set to 'Add' mode). The protocol is as follows, assuming the latest released hash-chain hash by the KDC is $H_{s,i-1}$:

Protocol 3. (Closed environment in the SFC)

$$
\begin{aligned}
w \quad &: r_w \xleftarrow{R} [1, n-1] \\
w \to * \; &: \mathsf{HELLO} \| r_w G \| N_w \\
s \to * \; &: \mathsf{ADD} \| [\mathsf{ADD}]_{H_{s,i}} \\
s \to * \; &: H_{s,i} \\
\forall v \in V_{\mathrm{SFC}} &\setminus \{w\}, \\
v \to s \; &: \{f(ID_v)\}_{K_{sv}} \| [\{f(ID_v)\}_{K_{sv}}]_{K_{sv}} \\
s \quad &: r_s \xleftarrow{R} [1, n-1] \\
&\quad K_{sw} \text{ is derived per Section 3(a)} \\
&\quad K'_M \xleftarrow{R} \mathcal{K}, (\kappa_w, KID_w) = CD(\mathcal{P}, w) \\
&\quad K'_G = [0]_{K_G} \\
&\quad \theta_1 = \mathsf{BOOT} \| r_s G \| N_s \| NID \| \{K'_M \\
&\quad \| \, \kappa_w \, \| KID_w \| K'_G\}_{K_{sw}} \| H_{s,i} \\
s \to w \; &: \theta_1 \| [N_w \| \theta_1]_{K_{sw}} \\
w \to s \; &: [N_s]_{K_{sw}} \\
s \quad &: u \xleftarrow{R} V_{\mathrm{SFC}} \setminus \{w\} \\
&\quad \theta_2 = \mathsf{DLGT} \| \{K'_M \| H_{s,i+1}\}_{K_{su}} \\
s \to u \; &: \theta_2 \| [\theta_2]_{K_{su}}
\end{aligned}
$$

The pairwise keys K_{sv} ($\forall v \in V_{\mathrm{SFC}} \setminus \{w\}$) are used to transport the key shares $f(ID_v)$ to the KDC. If there are less than τ operating nodes, the KDC would not be able to reconstruct the membership key K_M, and the KDC would not give w the necessary keying material to join the network. If there are enough key shares to reconstruct K_M, the KDC would verify if the hash of the reconstructed K_M is the same as the stored $H(K_M)$, and if verification succeeds, the KDC dispatches the necessary keying material to w.

u (randomly chosen) is delegated the task of broadcasting K'_M encrypted to existing members of the network. In order for u to do this, u must be able to relay the KDC's message in an authenticable fashion. Getting $H_{s,i+1}$ from the KDC allows u to do this.

Delegation to u After the nodes $\in V_{\mathrm{SFC}} \setminus \{w\}$ have been taken out of the SFC and returned to their original locations, u broadcasts a RENEW packet:

Protocol 4.

$$
\begin{aligned}
u \to *: \; &\mathsf{RENEW} \| \{K'_M\}_{K_G} \| [\{K'_M\}_{K_G}]_{H_{s,i+1}} \\
u \to *: \; &H_{s,i+1} \\
* \quad &: K'_G = [0]_{K_G}
\end{aligned}
$$

K'_M is encrypted with K_G so that only existing operating nodes can receive K'_M. Authentication is provided by $H_{s,i+1}$. All nodes refresh the global key as $K'_G = [0]_{K_G}$.

Neighbor discovery by w After the new node w has been taken out of the SFC and fixed at its intended location, it initiates neighbor discovery, by broadcasting a JOIN packet. It is possible that when w broadcasts its JOIN packet, w's neighbors have not received K'_M yet, so would ignore w's request. w would have to keep on trying until any of its neighbors respond, or until a certain retry limit is reached, depending on which event occurs first. A neighbor v, on hearing the w's JOIN packets in its operating phase instead of its deployment phase, would respond by unicasting

the necessary keying material to w. After neighbor discovery (delineated by a certain time-out), w becomes a regular member of the network. The protocol is as follows:

Protocol 5.

$w \to *:$ $\mathsf{JOIN}\|\{NID\|KID_w\}_{K'_M}$

$\forall v \in \mathcal{N}_w$,

v : $K_{vw} \leftarrow H(x)$ where $x \in \kappa_v \cap \kappa_w$

$\theta_1 = \{NID\|KID_v\}_{K'_M}\|\{K_v\}_{K_{vw}}\|H_{v,i_v}\|N_v$

$v \to w$: $\theta_1\|[\theta_1]_{K_{vw}}$

w : $K_{vw} \leftarrow H(x)$ where $x \in \kappa_v \cap \kappa_w$

$\theta_2 = \{K_w\}_{K_{vw}}\|H_{w,0}\|N_w$

$w \to v$: $\theta_2\|[N_v\|\theta_2]_{K_{vw}}$

$v \to w$: $[N_w]_{K_{vw}}$

After neighbor discovery (signalled by a certain time-out), every node generates deterministically a function $f(x)$ based on K'_M per Equation 1, stores $f(ID_v)$ and discards all the coefficients as well as K'_M. The old key share is also deleted.

PROPOSITION 1. *Suppose an attacker is within range of the BSN and hence has control over the air interface of the BSN throughout the operation lifetime of the BSN, but does not have physical access to any of the nodes. Suppose the hash chain $H_{s,i} = H(C_{i+1}\|H_{s,i+1})$ is used, where C_i is the counter corresponding to the ith epoch, and $|C_i| = c$. Provided $H : \{0,1\}^{c+m} \to \{0,1\}^m$ is an aPre-secure hash function that is indifferentiable from a random oracle, the attacker's advantage in adding a node to the BSN is at most $1-(1-2^{-m})^q$, where q is the number of calls of $H()$ made by the attacker.*

PROOF. By definition, an attacker successfully adds a node w to the network when w successfully establishes a secure channel with at least one of w's neighbors. However, w's neighbors would only respond to w's JOIN request after they have received a RENEW command. To achieve this, the attacker needs to forge a RENEW command, which requires the attacker to forge H_{i+1} (subscript s is dropped for simplicity) at the very least. The attacker's advantage in forging H_{i+1} is the probability of the attacker, with knowledge of the released keys H_0, H_1, ..., H_i, finding x such that $H(C_{i+1}|x) = H_i$. Formally, the attacker's advantage, using an algorithm A, is the conditional probability

$$\mathbf{Adv}(A) = \Pr\big[x \leftarrow A(H_i) : (x \ggg m) = C_{i+1} \wedge H(x) = H_i \big| H_{i-1} \leftarrow H(C_i|H_i) \wedge ... \wedge H_0 \leftarrow H(C_1|H_1)\big].$$

Provided that $H()$ is indifferentiable from a random oracle,

$$\mathbf{Adv}(A) = \Pr\big[x \leftarrow A(H_i) : (x \ggg m) = C_{i+1} \wedge H(x) = H_i\big].$$

Let us define algorithm A_0 as follows (where X is the domain, y is the target hash value, and q is the number of queries):

Algorithm $A_0(h,y,q)$
choose $X_0 \subseteq X$: $|X_0| = q \wedge (x \ggg m) = C_{i+1}, \forall x \in X_0$
foreach $x \in X_0$ { **if** $H(x) = y$ **then return** x }
return FAILURE

In the random oracle model, every $x \in X_0$ has a probability of 2^{-m} being mapped to y. The success probability of algorithm A_0 is therefore $\epsilon_{A_0} = 1 - \Pr[\text{no } x \text{ is mapped to } y] = 1-(1-2^{-m})^q$. In fact, algorithm A_0 can be shown to be the optimal algorithm, i.e., using any other algorithm A, $\epsilon_A \le \epsilon_{A_0}$. We prove this by induction on q. Let $q = 1$. A might start with a subset X_0 that does not contain only elements that have prefix C_{i+1}. Among the x's that have prefix C_{i+1}, each of them has a probability of 2^{-m} of being mapped to y, so $\epsilon_A \le \epsilon_{A_0}$ for $q = 1$. Suppose $\epsilon_A \le \epsilon_{A_0}$ for $q = k-1$. When $q = k$, $\epsilon_A = \Pr[\text{preimage not found in the previous } k-1 \text{ steps}]\Pr[\text{preimage found in the } k\text{th step}] + \Pr[\text{preimage found in the previous } k-1 \text{ steps}]$, i.e.,

$$\begin{aligned}\epsilon_A &\le (1-2^{-m})^{k-1}2^{-m} + 1 - (1-2^{-m})^{k-1} \\ &= 1-(1-2^{-m})^k\end{aligned}$$

Hence, $\epsilon_A \le \epsilon_{A_0}$ also holds for $q = k$. The above result can actually be obtained by comparing algorithm A_0 with algorithm FindPreimage [42] – they are the same except on how X_0 is chosen.

Algorithm FindPreimage(h,y,q)
choose $X_0 \subseteq X$ with $|X_0| = q$
foreach $x \in X_0$ { **if** $H(x) = y$ **then return** x }
return FAILURE

Therefore, $\mathbf{Adv}(A) \le 1-(1-2^{-m})^q$. □

10. NODE REMOVAL

When a node is removed from a BSN (detectable by time-out), *future key secrecy* (FKS) must be enforced, i.e., new group (global) keys must not be known by old members [41].

To remove a node, the only instruction a user has to follow is to switch off the node. All the keys are supposed to be stored in the RAM of the node, so once the node is switched off, all the keys are lost, and FKS is preserved.

What might happen is that an attacker might steal one or more of the nodes to read out the global key. Since there is no KDC *in* the network, there is no means to refresh the global key immediately, the strategy is to ensure each sensor monitor its neighbors' heartbeat/keep-alive packets. A node considers itself removed from the network when it stops receiving heartbeat packets for a certain duration. Once a node considers itself removed from the network, it erases all keys in its RAM. There are a few design considerations here:
(i) The heartbeat packets must be authenticated and fresh to prevent an attacker from forging heartbeat packets or replaying past heartbeat packets. This is readily achievable by using cluster hash chains.
(ii) This duration between the heartbeat packets of a node must be less than the time required by an attacker to successfully read out the keys in the RAM of the node, but more than the time required for putting τ nodes in an SFC. A conservative estimate in the order of a few minutes should probably be used, but more practical experience is needed to finetune the timing.

11. FORMAL VERIFICATION OF PROTOCOLS

While Proposition 1 gives the probability of an attacker adding a node to a BSN, it rules out active attacks like the planting of malicious nodes in the BSN that actively attack the protocols (notice the phrase "not have physical access" in Proposition 1). In the face of active attacks, we prove the security of Protocols 1 to 5 by formal verification. For

this, we use the automated tools ProVerif[3] and Scyther[4]. ProVerif is a theorem prover that represents a protocol by a set of Horn clauses. ProVerif supports unbounded number of sessions and unbounded message space. Scyther is a tool that uses symbolic analysis with backwards search based on partially ordered patterns (which represent infinite sets of traces). Scyther supports unbounded number of sessions but only guarantees termination for bounded number of sessions.

We use Scyther as our primary tool because (i) to specify simple protocols, its security protocol definition language is easier to use, and (ii) it has a convenient user interface. However, since Scyther does not support DH, we resort to ProVerif for Protocol 1 and 3. To simulate the SFC in Protocol 1 and Protocol 3, we declare a `private free` channel in ProVerif. To simulate DH, we use the `equation` construct:

```
fun dh/2. fun g/1. equation dh(x,g(y)) = dh(y,g(x)).
```

Using Scyther is rather straightforward for the other protocols. For each of the protocols, we verify that the secrecy of the relevant keys is maintained, and that mutual authentication property among the principals is achieved. Specifically, for Protocols 2 and 5, we have verified that even when K_M is compromised, the session key K_{uv} (or K_{vw}) is secure as long as the key shared by v and u (or w) is not compromised. Finally, since neither ProVerif and Scyther support timing mechanisms used in μTESLA, verification of Protocol 4 has not been performed. The scripts are available online[5].

12. CONCLUSION AND FUTURE WORK

KALwEN is a key management architecture for BSNs. It combines the cryptographic techniques of ECDH, combinatorial key pre-distribution, authenticated broadcast by one-way hash chains and threshold secret sharing in a complete framework. KALwEN addresses the usability, interoperability, hardware constraints and deployment issues of BSNs. In terms of usability, a user without expert knowledge needs but to follow a simple set of instructions to bootstrap or extend a network. Through user-friendly procedures, sensor devices from different manufacturers that expectedly do not have any pre-shared secret can establish secure communications with each other. KALwEN is lightweight enough to run on sensor node hardware, and is decentralized so that it does not depend on the availability of an LPU. While supporting global broadcast, local broadcast and neighbor-to-neighbor unicast, KALwEN is able to preserve past key secrecy and future key secrecy. The fact that all the cryptographic protocols of KALwEN have been formally verified also makes a convincing case.

One key future task is to improve upon current design of Protocol 1 such that it (i) requires less user involvement; and (ii) has less hardware requirements, so that for example a Faraday cage and public-key cryptography can be avoided. KALwEN at this stage is a proposal for addressing the constraints in Section 1; implementation is pending for substantiating the usability claim.

13. REFERENCES

[1] B. Alpern and F. B. Schneider. Key exchange using "keyless cryptography". *Information Processing Letters*, 1983.

[2] R. Anderson. A security policy model for clinical information systems. *IEEE Symposium on Security and Privacy*, pages 30–43, 1996.

[3] R. Anderson, H. Chan, and A. Perrig. Key infection: smart trust for smart dust. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP)*, pages 206–215, Oct. 2004.

[4] S.-D. Bao, Y.-T. Zhang, and L.-F. Shen. Physiological signal based entity authentication for body area sensor networks and mobile healthcare systems. In *27th Annual International Conference of the Engineering in Medicine and Biology Society (IEEE-EMBS 2005)*, pages 2455–2458, 2005.

[5] P. G. Bradford and O. V. Gavrylyako. Foundations of security for hash chains in ad hoc networks. *Cluster Computing*, 8(2):189–195, 2005.

[6] P. G. Bradford and O. V. Gavrylyako. Hash chains with diminishing ranges for sensors. *Int. J. High Performance Computing and Networking*, 4(1/2), 2006.

[7] I. Buhan, B. Boom, J. Doumen, P. Hartel, and R. Veldhuis. Secure Ad-hoc Pairing with Biometrics: SAfE. *International Journal of Security and Networks Special (IJSN) Special Issue on Secure Spontaneous Interaction*, 4(1), 2009.

[8] C. Castelluccia and P. Mutaf. Shake them up!: a movement-based pairing protocol for CPU-constrained devices. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 51–64. ACM, 2005.

[9] S. Çamtepe and B. Yener. Combinatorial design of key distribution mechanisms for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 15(2):346–358, Apr. 2007.

[10] Certicom Research. *Standards for Efficient Cryptography. SEC 1: Elliptic Curve Cryptography*, 1st edition, Sept. 2000.

[11] D. Chakrabarti, S. Maitra, and B. Roy. A key pre-distribution scheme for wireless sensor networks: merging blocks in combinatorial design. *International Journal of Information Security*, 5(2):105–114, Apr. 2006.

[12] S. Cherukuri, K. Venkatasubramanian, and S. Gupta. BioSec: a biometric based approach for securing communication in wireless networks of biosensors implanted in the human body. In *Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on*, pages 432–439, Oct. 2003.

[13] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *Advances in Cryptology – CRYPTO 2005, 25th Annual International Cryptology Conference*, volume 3621 of *LNCS*, pages 430–448. Springer-Verlag, 2005.

[14] J. Deng, C. Hartung, R. Han, and S. Mishra. A practical study of transitory master key establishment forwireless sensor networks. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, pages 289–302, Sept. 2005.

[15] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information*

[3] http://www.proverif.ens.fr

[4] http://people.inf.ethz.ch/cremersc/scyther

[5] http://yee.wei.law.googlepages.com

Theory, IT-22(6):644–654, 1976.

[16] W. Du, J. Deng, 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.

[17] L. Eschenauer and V. 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.

[18] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and Zero-Power defenses. In *29th IEEE Symposium on Security and Privacy*, pages 129–142, Oakland, California, May 2008. IEEE Computer Society.

[19] G. P. Hancke. Eavesdropping Attacks on High-Frequency RFID Tokens. In *Proceedings of the 4th Workshop on RFID Security (RFIDsec'08)*, pages 100–113, 2008.

[20] J.-H. Hoepman. Ephemeral pairing on anonymous networks. In *Security in Pervasive Computing*, volume 3450 of *LNCS*, pages 101–116. Springer-Verlag, 2005.

[21] C. Kuo, M. Luk, R. Negi, and A. Perrig. Message-in-a-bottle: user-friendly and secure key deployment for sensor nodes. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 233–246. ACM, 2007.

[22] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28:119–134, 2003.

[23] J. Lee and D. Stinson. A combinatorial approach to key predistribution for distributed sensor networks. In *IEEE Wireless Communications and Networking Conference*, volume 2, pages 1200–1205, Mar. 2005.

[24] A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*, pages 245–256. IEEE Computer Society, 2008.

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

[26] B. Lo and G. Yang. Key technical challenges and current implementations of body sensor networks. In *Proceedings of the Second International Workshop on Wearable and Implantable Body Sensor Networks*, pages 1–5, 2005.

[27] K. Malasri and L. Wang. Addressing security in medical sensor networks. In *HealthNet '07: Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments*, pages 7–12. ACM, 2007.

[28] K. Malasri and L. Wang. Design and implementation of a secure wireless mote-based medical sensor network. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 172–181, New York, NY, USA, 2008. ACM.

[29] R. Mayrhofer and H. Gellersen. Shake well before use: Authentication based on accelerometer data. In *Pervasive Computing*, volume 4480 of *LNCS*, pages 144–161. Springer-Verlag, 2007.

[30] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication. In *IEEE Symposium on Security and Privacy*, volume 0, pages 110–124, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[31] M. Moharrum, M. Eltoweissy, and R. Mukkamala. Dynamic combinatorial key management scheme for sensor networks. *Wireless Communications and Mobile Computing*, 6(7):1017–1035, 2006.

[32] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 28–37, New York, NY, USA, 2001. ACM.

[33] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. 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.

[34] C. Poon, Y.-T. Zhang, and S.-D. Bao. A novel biometrics method to secure wireless body area sensor networks for telemedicine and m-health. *IEEE Communications Magazine*, 44(4):73–81, Apr. 2006.

[35] P. Rogaway and T. Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *Fast Software Encryption*, volume 3017 of *LNCS*, pages 371–388. Springer-Verlag, 2004.

[36] D. Sánchez and H. Baldus. A deterministic pairwise key pre-distribution scheme for mobile sensor networks. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, pages 277–288, Sept. 2005.

[37] A. Seshadri, M. Luk, and A. Perrig. Sake: Software attestation for key establishment in sensor networks. In *Distributed Computing in Sensor Systems*, volume 5067 of *LNCS*, pages 372–285. Springer-Verlag, 2008.

[38] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure Code Update By Attestation in sensor networks. In *WiSe '06: Proceedings of the 5th ACM workshop on Wireless security*, pages 85–94. ACM, 2006.

[39] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[40] M. Shinagawa, M. Fukumoto, K. Ochiai, and H. Kyuragi. A near-field-sensing transceiver for intrabody communication based on the electrooptic effect. *IEEE Transactions on Instrumentation and Measurement*, 53(6):1533–1538, Dec. 2004.

[41] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A New Approach to Group Key Agreement. In *Proc. 18th Int. Conf. on Distributed Computing Systems*, pages 380–387, 1998.

[42] D. R. Stinson. Some observations on the theory of cryptographic hash functions. *Designs, Codes and Cryptography*, 38(2):259–277, 2006.

[43] A. Varshavsky, A. Scannell, A. LaMarca, and E. de Lara. Amigo: Proximity-Based Authentication

of Mobile Devices. In *UbiComp 2007: Ubiquitous Computing*, volume 4717 of *LNCS*, pages 253–270. Springer-Verlag, 2007.

[44] K. K. Venkatasubramanian, A. Banerjee, and S. K. S. Gupta. EKG-based key agreement in Body Sensor Networks. In *IEEE Conference on Computer Communications Workshops (INFOCOM)*, pages 1–6. IEEE, 2008.

[45] 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.