SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2011; **4**:1309–1329
Published online 3 December 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.256

RESEARCH ARTICLE

KALwEN: a new practical and interoperable key management scheme for body sensor networks

Yee Wei Law[1]*, Giorgi Moniava[2], Zheng Gong[3], Pieter Hartel[3] and Marimuthu Palaniswami[1]

[1] Department of Electrical & Electronic Engineering, The University of Melbourne, Parkville, VIC 3010, Australia
[2] School of Computer Science and Mathematics, Free University of Tbilisi, Bedia Street, Nutsubidze Plateau, I Micro District, Tbilisi 0183, Georgia
[3] Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, PO Box 217, 7500 AE, Enschede, The Netherlands

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 parameterized key management 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 secure global broadcast, local broadcast, and local (neighbor-to-neighbor) unicast, while preserving past key secrecy and future key secrecy (FKS). The fact that the cryptographic protocols of KALwEN have been formally verified also makes a convincing case. With both formal verification and experimental evaluation, our results should appeal to theorists and practitioners alike. Copyright © 2010 John Wiley & Sons, Ltd.

KEYWORDS
body sensor networks; body area networks; healthcare; security; key management

***Correspondence**
Yee Wei Law. Department of Electrical & Electronic Engineering, The University of Melbourne, Parkville, VIC 3010, Australia.
E-mail: ywlaw@unimelb.edu.au

1. INTRODUCTION

Specialized wireless sensor networks (WSNs) called body sensor networks (BSNs) are facilitating a revolution in the healthcare industry. A BSN is a wireless network of small, low-cost, biosensors *worn* (i.e., implantable sensors are excluded from this study as they require a different approach) by a human user, for the purpose of monitoring the user's physiological parameters, e.g., ECG, SpO_2, blood pressure etc [1].

The classic architecture of a BSN consists of a network of biosensors and an on-body *local processing unit* (LPU)—a device with a GUI and an input interface such as a PDA or mobile phone—directly or indirectly connected to a remote server storing the user's records [2]. An LPU is also simply called a controller by some researchers. 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.

The security and privacy problems related to healthcare systems are real [3]. As a recent study has demonstrated, medical devices that do not support any confidentiality and authentication function are prone to eavesdropping and attacks [4]. 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 constraints of a BSN. To properly motivate these constraints, we first review two user scenarios.

User Scenario 1 One of the classical applications for BSNs is real-time fall detection, because tripping is known to be one of the major causes of death among the elderly [5]. A typical system consists of multiple sensors embedded in the user's shoe soles. With existing technology, these sensors can detect after a fall event has happened, so as to alert the caregiver. In this case, a mobile phone connected to the sensors can be programmed to send a text to the caregiver in case of emergency. The ultimate goal however is to detect when a fall is about to happen, and prevent it from happening. A recent study shows that by applying vibratory stimuli to the sole, it is possible to stabilize the user [6]. In a system like this, vibratory actuators implement the countermeasure, rendering an LPU redundant. A subset of the security requirements is as such: communication from the sensors to the LPU should be authenticated, to prevent prank text messages to be sent; communication from the sensors to the actuators should be authenticated, to minimize discomfort for the user.

User Scenario 2 We give another user scenario from the Ambient Living with Embedded Network (ALwEN) project (https://www.alwen.nl). Patients who suffer from chronic obstructive pulmonary disease (COPD) often experience dyspnea (shortness of breath). For fear of triggering dyspnea by exercising, COPD patients are often trapped in a vicious cycle between lack of exercise and deteriorating health. A sensor system is being designed to measure a patient's ECG and breathing, and record his/her activity, with the primary objective of encouraging the patient exercise more without triggering dyspnea. To correlate the ECG and respiratory signal, time synchronization among the nodes is required. To prevent false alarm, the nodes need to exchange information to generate the proper context, e.g., 'the patient's heartbeat is accelerating in correlation with emotion', 'the patient's heartbeat is accelerating in correlation with exercise' etc.

Constraints and requirements In view of User Scenarios 1 and 2, here are the constraints and requirements that KALwEN is designed to address:

1. **Usability**: The key management process has to be autonomous enough so that barring some simple instructions a user has to follow, it requires no expert knowledge whatsoever of the user. It should be intuitive enough so that the user can operate the system in the comfort of his/her own home, without the need for a medical personnel or healthcare worker.
2. **Interoperability**: Sensor devices from different manufacturers should interoperate. For example, their hardware 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.
3. **Hardware**: BSNs have the same hardware constraints as WSNs do, i.e., limited computational power, limited memory, limited bandwidth, and limited energy. Moreover, a sensor node is typically not tamper-resistant.
4. **LPU**: When BSNs evolve from WSNs, they lose the dependability of a base station, because a BSN might not be connected to a base station at all times as the user moves about. In many designs, the LPU replaces the base station as the trusted third party, despite the outcome that the LPU becomes the single point of failure, and the fact that the LPU could be physically just as vulnerable as other nodes in the network. In our reference architecture, we relax the assumption of a base station and an LPU. This seemingly harsh constraint caters for all the foreseeable reasons why an LPU might not be available: most of the time it puts a weight on the user without achieving much; it is potentially costly; it could be hard to operate by an elderly. By not assuming either a base station or an LPU to be a permanent fixture of a BSN, we make it possible for KALwEN to be applicable to diverse network architectures, with the added benefit of being inter-manufacturer operable. To apply KALwEN to a network where an LPU is available, we treat the LPU as a regular node.
5. **In-network processing**: The main motivation for sensor networks is processing data in the network instead of collecting raw data. This is essential for context generation as evident in User Scenario 2, and for reducing usage of bandwidth, energy, and storage.

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 equipment to diagnose the user, so emergency accessibility of a BSN is less critical. Since we are not addressing implantable sensors, any sensor that 'gets in the way' can just be switched off and set aside.

Contribution Our objective is to propose a key management scheme under the above constraints. Our contribution is KALwEN (short for Key management for ALwEN). KALwEN combines the most relevant techniques in the literature in a complete framework. KALwEN satisfies the above-mentioned constraints or requirements, and supports the three basic communication modes: global broadcast, local broadcast, and local unicast, while preserving past key secrecy and future key secrecy (FKS). Using formal verification, we show that the cryptographic protocols of KALwEN are secure. KALwEN is an unconventional design with the primary purpose of creating a level playing field for body sensor vendors, in which devices of different manufacturers can interoperate; and the secondary purpose of freeing the users from the burden of operating a computer and carrying an LPU.

Organization 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 based on which KALwEN is built. Sections 6– 10 specify the protocols for the factory phase, bootstrap phase, deployment phase, and operation phase of a BSN. Sections 11 and 12 present our formal verification and experimental results, respectively. In Section 13, we discuss in detail the merits of KALwEN compared to some existing schemes, and attempt to dissolve some common misunderstanding and criticism of KALwEN. Finally, Section 14 concludes the paper.

2. RELATED WORK

BSNs, being descended from WSNs, can benefit from the many ideas that have already been proposed for WSNs. In fact, key management ideas can be borrowed from many other areas. Our discussion is divided into the following subsections: WSNs, ubiquitous computing, and healthcare sensor networks.

2.1. Wireless sensor networks

Perrig *et al.* [7] pioneer the use of one-way hash chains in the form of μTESLA to achieve authenticated broadcast in WSNs. Anderson *et al.* [8] suggest that during the bootstrapping phase of a network, keys can be exchanged in the clear. LEAP+ [9] and its improved variants [10] 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.* [11] propose establishing a pairwise key between a base station and each node in a Faraday cage. However, for bootstrapping multiple nodes at the same time, the nodes are required to be of the same type. Kuo *et al.* also recommend using software attestation [12] 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 agreement, software attestation can be used for key establishment [13]. 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. Castellucia *et al.* describe more pitfalls of software attestation in a recent work [14].

The random key pre-distribution paradigm is pioneered by Eschenauer *et al.* [15] and later extended by Liu *et al.* [16] and Du *et al.* [17]. The idea is to prepare a pool of 'keying material' (which can be single symmetric keys [15], polynomials [16], or matrices [17]), 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 [18–20]. 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 (if the number of nodes n is small, a node can afford to store $n - 1$ keys, one for every other node), combinatorial key pre-distribution seems to be the direction forward for WSNs, and by extension BSNs.

2.2. Ubiquitous computing

Ideas from ubiquitous computing can also be useful. The literature of this area primarily focuses on *secure pairing* [21]. 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. Saxena *et al.*'s 'Blink 'Em All' [22] uses the visual channel as the band-limited side channel and the wireless channel as the insecure public channel. The wireless channel is used for transmitting initial commitments whereas the visual channel is used for transmitting *short authenticated strings*, as part of the protocol designed by Laur *et al.* [23]. 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 [24,25]. Another paradigm is to derive a common key by sampling similar side channels. For example, co-located devices experience similar radio environments [26]; devices shaken together can establish a common key [27]. Other examples that are based on this paradigm involve a pair of human users exchanging visual information, but these are not as useful [28,29] 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.* [30] for BSNs. Later work investigates the feasibility of using the heart rate variability [31], the interval pulse [32], the electrocardiogram [33,34] 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.

2.3. Healthcare sensor networks

Complete key management schemes have only started appearing in recent years. Blinking LEDs Indicated Group (BLIG) is an approach where nodes exhibit a synchronized blinking pattern when they are securely grouped [35]. BLIG is different from Blink 'Em All, in that the former uses public-key cryptography for key establishment and the visual channel only for verification by a human user that the protocol completes successfully [36], whereas the latter uses the visual channel for signaling short strings. BLIG has been incorporated by Keoh *et al.* [37] and Li *et al.* [38] into their protocols. In Section 13, we provide a detailed comparison between BLIG, Keoh *et al.*'s scheme [37], Li *et al.*'s scheme [38], and KALwEN.

Some key management architectures rely on a local base station to authenticate sensor nodes biometrically [39,40]. In another architecture [41], secure interaction between 'patient security processors', 'clinician security processors', 'nurse security processors', and a central server is facilitated by a key establishment scheme. 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 I partially summarizes the symbols used in this paper.

3.1. 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 [42]. 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 [43, Section 6.1] using the elliptic curve cofactor Diffie–Hellman primitive (which is resistant to small subgroup attacks compared to the original primitive) [43, 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 computationally 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* [44].

An elliptic curve cryptosystem is built on a set of *domain parameters* denoted by (q, FR, a, b, G, n, h) (see Table I for explanation). 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 computes $(x, y) = hd_ud_vG$ and v computes the same point; (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 [45].

Table I. Partial list of symbols.

$\{\cdot\}_K$	Encryption function using key K
$[\cdot]_K$	Message authentication function using key K
$H(\cdot)$	Cryptographic hash function
$\|$	Concatenation operator
$\overset{R}{\leftarrow}$	Uniformly chosen at random from
$\ggg$	Right shift operator
V_{SFC}	Set of all nodes excluding the key distribution center 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 of an elliptic curve: q is the order of the finite field on which the curve is defined; FR is the field representation; a and b are the coefficients of the curve; G is the base point on the curve; n is a prime of order G; h is the number of rational points on the curve divided by n
$\mathcal{K}$	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'	Renewed version of key K
N_v	Nonce sent by v
τ	The least number of nodes removed that will alert the user; 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

3.2. 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, to ensure every pair of neighbors can establish a pariwise key. While this type of schemes are in general less resilient to node capture attacks than random key pre-distribution schemes, these schemes are more user-friendly because they incur less communication overhead at deployment. Furthermore, an attacker cannot capture too many nodes without alarming the user. Therefore high probability of key-share 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 [46], but the resilience of this scheme is unsatisfactory (capturing a node compromises half of the key pool). A better approach is *symmetric designs* [20]. 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, which is more than the size expected of a typical 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 [20], 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 [47] (an implementation of the polynomial-based variant for BSNs has been reported [48]), 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.

3.3. One-way hash chain

The one-way hash chain [49] is a lightweight replacement for asymmetric-key digital signatures. To bootstrap the protocol, the sender s first generates a one-way hash chain $\{H_{s,l-1}, H_{s,l-2}, ..., H_{s,0}\}$, where $H_{s,i} = H(H_{s,i+1})$ $(i = 0, \ldots, l-2)$, $H()$ 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. To send message M_i to the receivers, the sender broadcasts $M_i \| [M_i]_{H_{s,i}}$. When the sender is sure all the receivers have received the message (which is trivial to achieve in a single-hop network), the sender discloses $H_{s,i}$. 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 if (ii) $H_{s,i}$ generates $[M_i]_{H_{s,i}}$.

The one-way hash chain is provably secure if the hash function is l-wise independent [50]. However, 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 l is large. In practice, a larger domain for the hash function should be used; for example, by coupling the hash chain with a chain of salts [51], i.e., $H_{s,i} = H(\text{salt}_{s,i+1} \| H_{s,i+1})$. Meanwhile, Bradford *et al.* [52] propose using a separate chain of counters which do not need to be transmitted; they also propose $H_{s,i} = H(H_{s,i+1} \| H_{s,i+2} \| \ldots)$. Throughout our ensuing discussion, for clarity, we continue to write the one-way hash chain in its original form, with the implicit understanding that in implementation, the hash chain is coupled with a chain of salts or counters. Moreover, to facilitate the proof of Proposition 1, the hash function must also be *always preimage-resistant* (aPre-secure [53]) and indifferentiable from a random oracle [54].

3.4. 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 [55]. 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 $f(ID_i)$ $(i = 1, \ldots, n)$, 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 (FSM). At the end of this section, we also describe the equipment that is needed to support the nodes.

KALwEN's mission is to support authentication in these communication modes: (i) local unicast, (ii) local broadcast, (iii) global broadcast. The first two communication modes are the basic operations of a medium access protocol. Local broadcast is also a primitive for in-network processing. Global broadcast is necessary for announcing network-wide events. KALwEN supports the communication modes with the following key types:

- Confidential and authenticated local unicast by the nodes: *pairwise key*
- Confidential and authenticated local broadcast by the nodes: *global key* and *cluster hash chain*
- Confidential and authenticated global broadcast by the key distribution center (KDC): *global key* and *global hash chain*

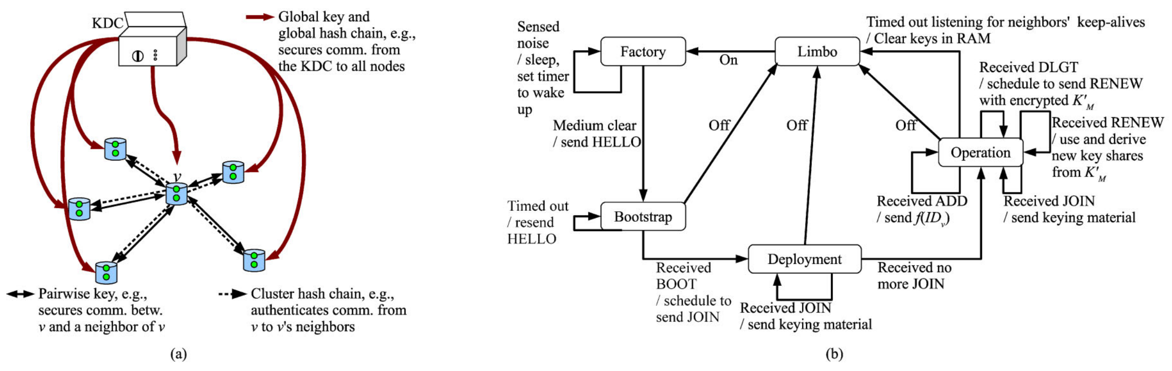

KDC
Global key and global hash chain, e.g., secures comm. from the KDC to all nodes
v
Pairwise key, e.g., secures comm. betw. v and a neighbor of v
Cluster hash chain, e.g., authenticates comm. from v to v's neighbors
(a)
Sensed noise / sleep, set timer to wake up
Factory
On
Limbo
Timed out listening for neighbors' keep-alives / Clear keys in RAM
Received DLGT / schedule to send RENEW with encrypted K'_M
Medium clear / send HELLO
Off
Off
Off
Received RENEW / use and derive new key shares from K'_M
Operation
Timed out / resend HELLO
Bootstrap
Received ADD / send f(ID_v)
Received JOIN / send keying material
Deployment
Received BOOT / schedule to send JOIN
Received JOIN / send keying material
Received no more JOIN
(b)

Figure 1. (a) Essential key types; (b) simplified finite state machine of a node connecting the different phases.

A pairwise key is shared between two neighboring nodes. The global key is shared among all the nodes. The commitment of a cluster hash chain is shared by a node with its neighbors. The commitment of the global hash chain is shared by the KDC with all other nodes. The KDC will be discussed later. Figure 1(a) illustrates the distribution of these keys in a network.

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 environment, 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 switched on, its boot loader 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. All ephemeral cryptographic keys are stored in the RAM, so that when a node is switched off, all keys are lost. To reduce accidental switch-ons/offs, a mechanism should be in place that requires a certain amount of cognitive effort from the user to switch on/off the device. An analogy for this mechanism is readily found on most of today's mobile phones (switching on a phone requires the hang-up button to be pressed for a few seconds).

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

Equipment To bootstrap a network, and to add a node to a network, an additional equipment is needed: a 'Smart Faraday Cage' (SFC). A Faraday cage is a metal enclosure for containing the electromagnetic fields of the equipment within. A Faraday cage, although not yet commercially available for BSNs, can already be purchased for individual mobile devices.[†] 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 the following components:

1. 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 before going to sleep; (iii) 'Add' for bootstrapping nodes to be added to the previously bootstrapped network;
2. an indicator showing one of the three states: 'Working', 'Done', and 'Error';
3. and a jammer, attached to the exterior of the SFC, that constantly transmits a noise signal when the knob points to 'Bootstrap' or 'Add'.

One security requirement of the SFC is that it must be dimensioned such that any two nodes in it are within the range of each other. The functional requirements of the SFC are defined by the protocols the KDC has to follow, as described in the following sections. *It is to be emphasized that an SFC is only needed when bootstrapping a network or adding a node to a network.* The SFC is a personal device and is only used to bootstrap the owner's sensor nodes.

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.

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

Due to the small scale of the human body, a BSN is fully connected. There exists a range around a BSN outside which no attacker can eavesdrop on the messages of the BSN, even with advanced skills and equipment [56].[‡] When an attacker is in range, the attacker can forge, intercept, and arbitrarily manipulate any message, as dictated by the Dolev–Yao model [57]. When an attacker is out of range, the attacker can only forge messages.

The above are standard assumptions. Below we list assumptions specific to KALwEN:

Assumption 1 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 security of a single component rather than a host of smaller components that are far less physically manageable.

Assumption 2 The time to read cryptographic keys out of the RAM of a sensor node (in the order of hours) is longer than the interval between the keep-alive packets of a network (in the order of minutes). We note that it is demonstrably possible to extract keys from a Crossbow MICA2 node in under one minute [58], but MICA2 is mostly a research-purpose device with the Joint Test Action Group (JTAG) interface fully exposed. Removing the JTAG interface, using ball grid array chips for the memory components [59, p.293], applying a layer of epoxy around the memory components are just some of the low-cost measures to make a production device considerably harder to tamper with.

Assumption 3 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. At the same time, a node in a BSN has at least $\tau - 1$ neighbors, i.e., the network degree $\geq \tau - 1$. This assumption should be realistic because τ is typically quite low (it is not easy to ignore a few nodes missing from one's body).

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 3.1).

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 built-in pseudorandom number generators.

All of the above also applies to SFCs.

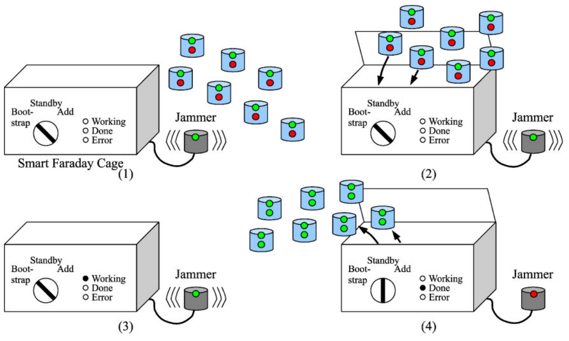


Figure 2. Procedure for bootstrapping nodes.

7. BOOTSTRAP PHASE

The bootstrap process takes place in a controlled environment. Apart from the nodes themselves, the user needs an SFC. The instructions a user has to follow are (see Figure 2):

1. set the SFC to 'Bootstrap', then switch on or reset the nodes in close proximity of the SFC;
2. put the nodes in the SFC;
3. seal the SFC;
4. wait for the 'Done' indicator and then deploy the nodes.

The technical detail behind the instructions is given 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 goes 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. It is clear at this stage that the purpose of the jammer is two-fold: to prevent the nodes from initiating bootstrap outside the SFC, and to prevent residual radiation from within the SFC from being useful to any potential eavesdropper.

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 $|V_{\text{SFC}}|$ 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 allocates 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:

[‡] Hancke shows that the forward channel of ISO 14443A and ISO 15693 near-field communication can be eavesdropped as far as 10 m away, even though these systems are advertised to operate within 10 cm. It can be safely assumed that commodity far-field communication can be eavesdropped at more than 10 m away.

Protocol 1. (Closed environment in the SFC)

$$
\begin{aligned}
&\forall_v \in V_{\text{SFC}},\\
&v \quad : r_v \overset{R}{\leftarrow} [1, n-1]\\
&v \to * : ID_v \| ID_* \| \text{HELLO} \| r_v G \| N_v\\
&s \quad : r_s \overset{R}{\leftarrow} [1, n-1]\\
&\qquad K_{sv} \text{ is derived per Section 3.1}\\
&\qquad K_M \overset{R}{\leftarrow} \mathcal{K} \text{ (detail later)}\\
&\qquad (\kappa_v, KID_v) = CD(\mathcal{P}, ID_v)\\
&\qquad K_G \overset{R}{\leftarrow} \mathcal{K},\ H_{s,l-1} \overset{R}{\leftarrow} \{0,1\}^m\\
&\qquad H_{s,i-1} = H(H_{s,i}) \forall i \in \{1, \ldots, l-1\}\\
&\qquad \theta = ID_s \| ID_v \| \text{BOOT} \| r_s G \| N_s \| NID \| \tau\\
&\qquad \| \{K_M \| \kappa_v \| KID_v \| K_G\}_{K_{sv}} \| H_{s,0}\\
&s \to v : \theta \| [\theta \| N_v]_{K_{sv}}\\
&v \to s : ID_v \| ID_s \| [N_s]_{K_{sv}}
\end{aligned}
$$

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 $s \to v$), 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 convention [7]. 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 $H(\text{key}\|\text{message})$ *if* H can be modeled as a random oracle [60].

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 the commitment of its cluster hash chain to all its neighbors, per Protocol 2.

Protocol 2.

$$
\begin{aligned}
&\forall_u \in V,\\
&\quad u \to * : \text{JOIN} \| \{NID \| KID_u\}_{K_M}\\
&\forall_v \in \mathcal{N}_u,\\
&\quad v \quad : K_{uv} \leftarrow H(x) \text{ where } x \in \kappa_u \cap \kappa_v\\
&\qquad\quad \theta_1 = KID_v \| H_{v,0} \| N_v\\
&\quad v \to u : \theta_1 \| [\theta_1]_{K_{vu}}\\
&\quad u \quad : K_{uv} \leftarrow H(x) \text{ where } x \in \kappa_u \cap \kappa_v\\
&\quad u \to v : H_{u,0} \| [H_{u,0} \| N_v]_{K_{uv}}
\end{aligned}
$$

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 timeout), every node generates *deterministically* a function

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

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 $(i = 0, \ldots, \tau - 1)$. 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 unicast and secure local broadcast. Each node can also authenticate broadcast from the KDC, when the broadcast is relayed by a delegate 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 [61]. 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 again. 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.

To add a node to the network, a user has to follow these instructions (Figure 3):

1. set the SFC to 'Add', then switch on or reset the new node in close proximity of the SFC;
2. put the new node, and any τ of the operating nodes into the SFC;
3. seal the SFC;
4. wait for the 'Done' indicator and then deploy the nodes, with the old nodes first and the new node last.

The technical detail behind the instructions is given below.

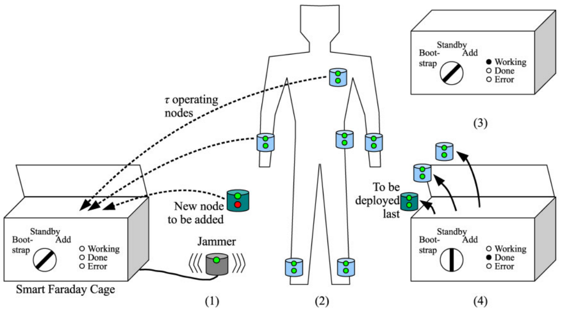


Figure 3. Procedure for adding a node where $\tau = 2$.

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)

$$\begin{aligned}
w \quad &: r_w \overset{R}{\leftarrow} [1, n-1] \\
w \to * &: \text{HELLO} \| r_w G \| N_w \\
s \to * &: \text{ADD} \| [\text{ADD}]_{H_{s,i}} \\
s \to * &: H_{s,i} \\
\forall_v \in V_{\text{SFC}} \backslash \{w\}&, \\
v \to s &: \{f(ID_v)\}_{K_{sv}} \| [\{f(ID_v)\}_{K_{sv}}]_{K_{sv}} \\
v \quad &: r_s \overset{R}{\leftarrow} [1, n-1] \\
&\quad K_{sv} \text{ is derived per Section 3.1} \\
&\quad K'_M \overset{R}{\leftarrow} \mathcal{K}, (\kappa_w, KID_w) = CD(\mathcal{P}, ID_w) \\
&\quad K'_G = [0]_{K_G} \\
&\quad \theta_1 = \text{BOOT} \| r_s G \| N_s \| NID \| \ \tau \\
&\quad \| \{K'_M \| \kappa_w \| KID_w \| K'_G\}_{K_{sw}} \| H_{s,i} \\
s \to w &: \theta_1 \| [\theta_1 \| N_w]_{K_{sw}} \\
w \to s &: [N_s]_{K_{sw}} \\
s \quad &: u \overset{R}{\leftarrow} V_{\text{SFC}} \backslash \{w\} \\
&\quad \theta_2 = \text{DLGT} \| \{K'_M \| H_{s,i+1}\}_{K_{su}} \\
s \to u &: \theta_2 \| [\theta_2]_{K_{su}}
\end{aligned}$$

The pairwise keys K_{sv} ($\forall v \in V_{\text{SFC}} \setminus \{w\}$) 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. This delegation is necessary because the SFC would not be around when the nodes are deployed later. 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 $\in V_{\text{SFC}} \setminus \{w\}$ have been taken out of the SFC and returned to their original locations, u broadcasts a RENEW packet:
Protocol 4.

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

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.

$$\begin{aligned} w \to * &: \text{JOIN}\|\{NID\|KID_w\}_{K'_m} \\ \forall_v \in \mathcal{N}_w&, \\ v &: K_{vw} \leftarrow H(x) \text{ where } x \in \kappa_v \cap \kappa_w \\ &\quad \theta_1 = KID_v\|H_{v,i_v}\|N_v \\ v \to w &: \theta_1\|[\theta_1]_{K_{vw}} \\ w &: K_{vw} \leftarrow H(x) \text{ where } x \in \kappa_v \cap \kappa_w \\ w \to v &: H_{w,0}\|[H_{w,0}\|N_v]_{K_{vw}} \end{aligned}$$

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 hash, and $|C_i| = c$. Provided $H : \{0,1\}^{c+m} \to \{0,1\}^m$ is a 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 $H_0, H_1, \ldots, H_i$, finding x such that $H(C_{i+1}\|x) = H_i$. Formally, the attacker's advantage, using an algorithm A, is the conditional probability

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

Note: $\wedge$ represents logical AND, not intersection. Provided that $H()$ is indifferentiable from a random oracle,

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

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, $\epsilon_A \leq \epsilon_{A_0}$. 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 $\epsilon_A \leq \epsilon_{A_0}$ for $q = 1$. Suppose $\epsilon_A \leq \epsilon_{A_0}$ 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.,

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

Hence, $\epsilon_A \leq \epsilon_{A_0}$ also holds for $q = k$. The above result can actually be obtained by comparing algorithm A_0 with algorithm FindPreimage [62]—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, $\mathbf{Adv}(A) \leq 1 - (1 - 2^{-m})^q$. ∎

10. NODE REMOVAL

When a node is removed from a BSN (detectable by timeout), *future key secrecy* (FKS) must be enforced, i.e., new group (global) keys must not be known by old members [61].

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. As mentioned, to reduce accidental switch-offs, a mechanism should be in place that requires a certain amount of cognitive effort from the user to switch off the device.

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' keep-alive packets. A node considers itself removed from the network when it received at most $\tau - 2$ keep-alive packets for a keep-alive interval. Consider $\tau \geq 2$ for now; $\tau = 1$ will be considered as a special case later. Once a node considers itself removed from the network, it erases all keys in its RAM. The rationale for this algorithm is as follows. If a node is removed from a network together with at most $\tau - 2$ neighbors, then the node will receive at most $\tau - 2$ keep-alives per keep-alive interval, and the node will consider itself removed from the network. If a node is removed from a network together with at least $\tau - 1$ neighbors, or in other words, at least τ nodes have been removed from the network, then by Assumption 3, the user will become aware of the attack attempt. Note the special case of $\tau = 1$: a node does not check for keep-alive packets, because by Assumption 3, a user is aware of the theft of even a single node.

There are a few design considerations regarding this algorithm:

- Keep-alive packets must be authenticated and fresh to prevent an attacker from forging keep-alive packets or replaying past keep-alive packets. This is readily achievable by using cluster hash chains.
- A keep-alive interval 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. Per Assumption 2, a keep-alive interval can be as high as 60 min, but more experience with the users and the actual hardware is needed to finetune the timing for the optimal tradeoff between usability and security.

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[§] and Scyther.[‖] 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 the delayed preimage disclosure mechanism used in the one-way hash chain, verification of Protocol 4 has not been performed. The ProVerif script for Protocol 1 is provided in the Appendix. The Scyther scripts for the other protocols are available on the first author's home page.[¶]

12. EXPERIMENTAL RESULTS

The experiments are carried out on two major hardware platforms: Crossbow TelosB and Crossbow IRIS running TinyOS 2.x to evaluate the practicality of KALwEN. These two platforms are chosen because of their opposing characteristics: TelosB has more RAM than IRIS (10 KB *vs.* 8 KB) but IRIS has a larger Flash memory than TelosB

[§] http://www.proverif.ens.fr

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

[¶] http://sites.google.com/site/yeeweilaw

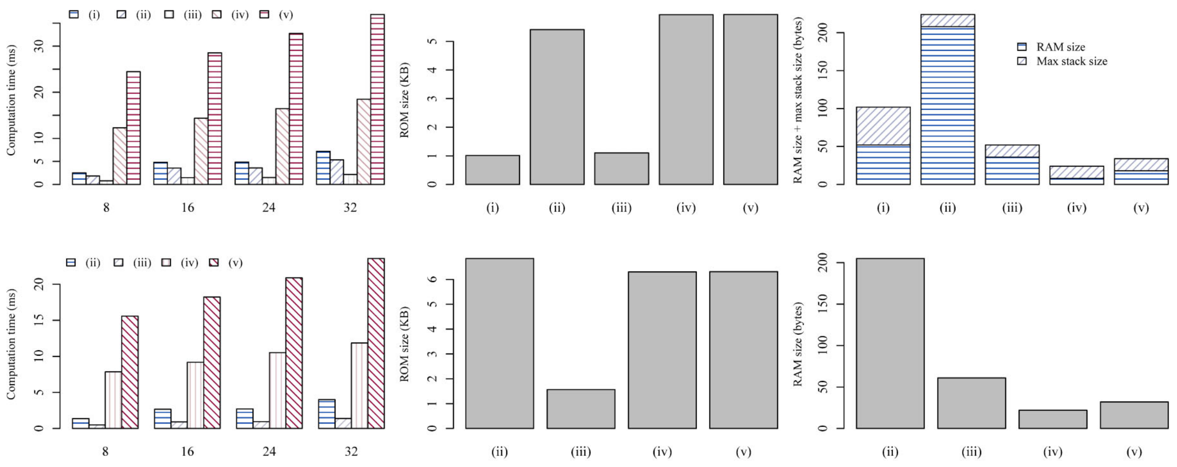


Figure 4. Performance and resource usage of (i) hardware AES-CBC-MAC, (ii) software AES-CBC-MAC, (iii) Skipjack-CBC-MAC, (iv) TuLP, (v) TuLP-128 on TelosB (top row), and IRIS (bottom row). *Note*: IRIS does not support hardware AES-CBC-MAC; stack size information is not available for IRIS.

(128 KB *vs*. 48 KB); TelosB's transceiver CC2420 supports hardware AES encryption but IRIS does not. Our IRIS implementation uses the Flash memory as much as possible, for example, the tables of a block cipher are all stored in the Flash memory.

We benchmark the bootstrap protocol (Protocol 1) because it involves the most public-key cryptographic operations. We first choose the key size. For a *security margin* of 2012, a symmetric key of at least 80 bits should be used [63]. The definition of security margin here follows that of Lenstra and Verheul [63]: suppose (1) c_{DES} is the number of computations required to break DES, (2) c_X is the number of computations required to break algorithm X, and (3) an attacker that can afford c_{DES} computations starting from year 1982 can afford c_X computations starting from year y, then the security margin of algorithm X is y. Corresponding to a security margin of 2012, an ECC key bit-length of between 149 and 165 should be used [63]. So we pick 160-bit ECC keys.

In the benchmarking results below, by convention, we write ROM size to refer to the size of the `text` (program code) and `data` (initialized data) segments; and RAM size to refer to the size of the `data` and `bss` (uninitialized data) segments. Maximum run-time stack size on the TelosB, obtained using MSPSim, is also provided. There is currently no instruction-level simulator for measuring run-time stack usage on the IRIS.

Block cipher 80-bit cipher Skipjack has been chosen by Karlof *et al*. [64], and by Law *et al*. [65] for applications with low security requirements. However, TelosB supports hardware AES encryption, so we use AES for TelosB and expand the block cipher key size to 128 bits on TelosB. On IRIS, Skipjack is used. For hardware-accelerated AES, we use Zhu's code[#] whereas for Skipjack we use Law *et al*.'s code [65], which uses less RAM than the implementation in TinySec [64].

MAC The natural choice for a MAC to be used with a block cipher is CBC-MAC because it is simple and provably secure [66]. Our implementation of CBC-MAC follows the 'length prepending' variant [66, p.395]. Figure 4 compares AES-CBC-MAC, Skipjack-CBC-MAC, TuLP and TuLP-128 [67]. TuLP and its more secure variant TuLP-128 are two MAC algorithms proposed recently for resource-constrained devices [67]. TuLP and TuLP-128 have similar code size as AES, but significantly lower RAM usage than AES. Performance-wise however, TuLP and TuLP-128 fare worse than AES-CBC-MAC and Skipjack-CBC-MAC. Gong *et al*.'s results suggest that TuLP and TuLP-128 are more suitable for hardware implementation [67]. Of particular interest is that on TelosB, hardware-accelerated AES-CBC-MAC is slower than software AES-CBC-MAC and Skipjack-CBC-MAC. As the implementation only makes use of the internals of TinyOS, namely the components ActiveMessageC, HplCC2420PinsC, CC2420SpiC, CC2420SpiWireC, the performance bottleneck of the hardware AES routine is thought to lie amidst one or some of these components. We are aware of an article that reports a better performance result for CC2420-accelerated AES [68] but the authors are not able to provide their code. We opt to use hardware-accelerated AES-CBC-MAC on TelosB anyway, in the hope that this performance bottleneck will be removed in the future. On IRIS, Skipjack-CBC-MAC is the clear all-around winner.

ECDH To implement ECDH, we use the publicly available TinyECC [44]. We are aware of more optimized ECC implementations, such as NanoECC [69] and Driessen *et al*.'s [70], but they are not publicly available. We choose the domain parameters secp160r1 [71]. According to Table II, secp160k1 offers the best code size; secp160r2 is similar to secp160k1 in speed but worse than secp160k1 in size; secp160r1 offers the best key agreement time. We choose secp160r1 for our experiments. Many researchers

[#] http://cis.sjtu.edu.cn/index.php/The_Standalone_AES_Encryption_of_CC2420_(TinyOS_2.10_and_MICAz)

Table II. Comparison of domain parameters in terms of (i) the initialization time of ECDH in seconds, (ii) the ECDH key agreement time in seconds, (iii) the ROM size of the domain parameters in bytes, (iv) the maximum run-time stack size of ECDH in bytes, (v) the ROM size of ECDH excluding the domain parameters in bytes, and (vi) the RAM size of ECDH in bytes. Domain parameters by themselves use 0 RAM.

	(i)	(ii)	(iii)	(iv)	(v)	(vi)
			TelosB			
secp160k1	2.62	2.78	530	750	11166	1866
secp160r1	2.61	2.66	532	740		
secp160r2	2.60	2.79	544	750		
			IRIS			
secp160k1	1.88	2.10	492	N/A	14480	1774
secp160r1	1.86	1.76	776	N/A		
secp160r2	1.87	2.12	576	N/A		

choose secp160r1 by default but to our knowledge, this is the first time the selection is justified performance-wise. When using TinyECC, all the optimizations are enabled for reasonable performance in human terms (seconds are good, tens of seconds are too long).

Hash function On TelosB, it is possible to take advantage of the hardware AES to construct a hardware-accelerated hash function using the Davies-Meyer construction [72], that could be faster [73] than SHA-1. This approach is employed by Andersen [36]. However, SHA-1 is the only supported hash function in the ECC standard [43], so we use SHA-1 as the hash function.

Network bootstrapping For benchmarking the bootstrap protocol (Protocol 1), up to 12 regular nodes are first switched on, followed by an additional node acting as the SFC. We measure the time between when the first HELLO packet arrives at the SFC and when the last BOOT packet arrives at a regular node. Note that the acknowledgement packets on the last line of Protocol 1 are not taken into account because these packets can be piggybacked on the suitable messages in Protocol 3 (on line 6 to be exact). The results are in Figure 5. The optimal case is when the SFC bootstraps each node right after another. When this happens, ignoring computation time *except for* ECDH key agreement, and ignoring communication times (backoff timers, transmission latencies etc.), the time to bootstrap a network of N nodes is $2.66N$ for TelosB, and is $1.76N$ for IRIS (we get the constants 2.66 and 1.76 from Table II). Figure 5 shows that the times to bootstrap nodes are reasonably close to the optima. For example, the time to bootstrap 12 nodes is a little under 25 s, which should be acceptable to most users. Clearly, the bootstrap latency can be further improved by using a more capable processor or an ECC accelerator for the SFC (an ECC processor has even become feasible on an RFID tag nowadays [74]).

Network deployment During deployment, the nodes perform neighbor discovery to establish pairwise keys and distribute the commitment of their cluster hash chain to their neighbors (Protocol 2). In the protocol, each node

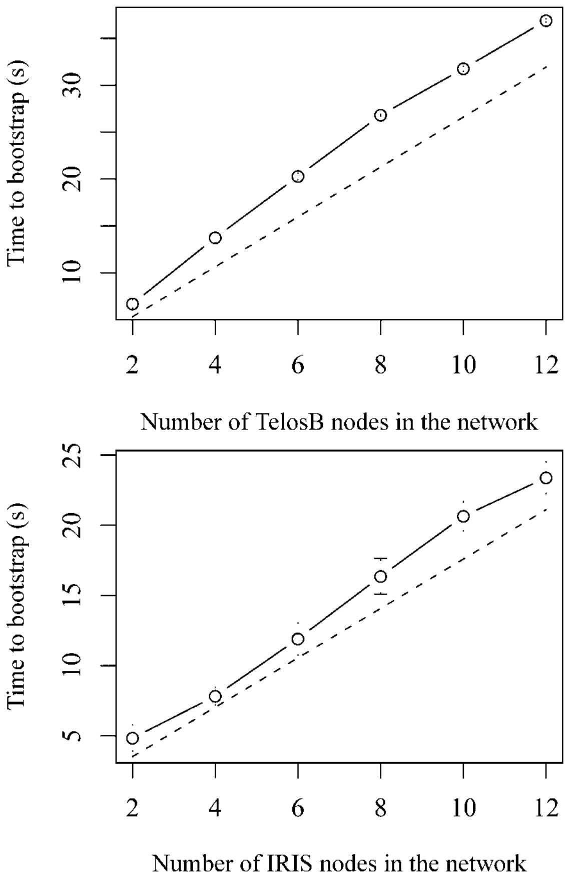
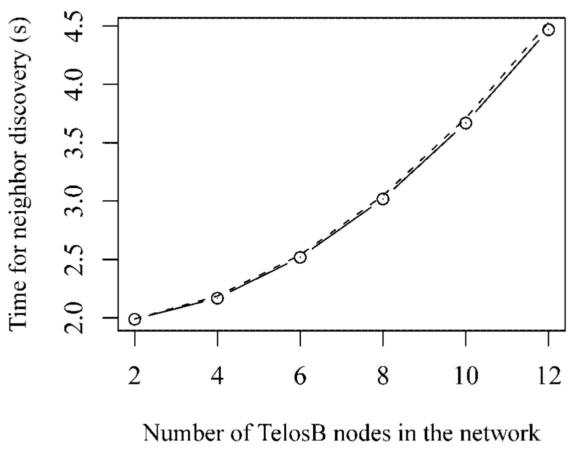


Figure 5. Timing figures for the bootstrap protocol (Protocol 1) on TelosB and IRIS. Dashed lines represent the optima.

sends $O(n-1)$ messages, so the total number of messages is $O(n(n-1))$. In our implementation, at the start of deployment, every node sets a time-out of 2000 clock ticks. Before the time-out, every node contends for the medium to send its JOIN packet. At time-out, every node knows how many neighbors it has based on the JOIN packets it received. It sorts the neighbor IDs, and determines the position of its ID among the sorted neighbor IDs. Say its ID ranks at the ith place among the neighbor IDs, then it will schedule to transmit its replies to JOIN requests $(i-1)(n-1) \times 20$ ticks later. Note that there are $(n-1)$ JOIN requests to reply to, and 20 ticks are allocated to the transmission of each reply. Using this algorithm, the time required to deploy n nodes is

$$(2000 + n(n-1) \times 20)/1024\,\text{s} \tag{2}$$

Figure 6 shows the experimental times required to deploy up to 12 nodes, which are close to the prediction using (2). Although the time is quadratic in n, taking under 4.5 s to deploy 12 nodes seems reasonable.



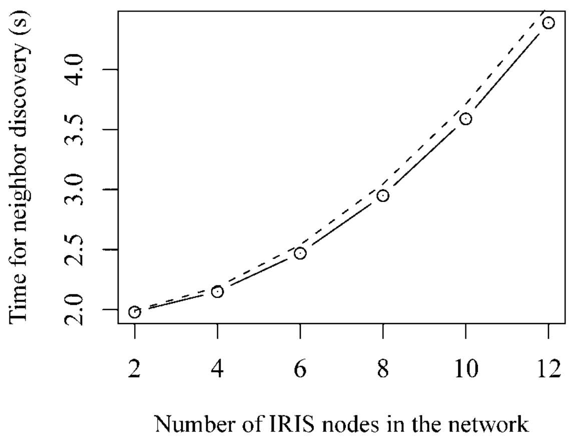


Figure 6. Timing figures for the neighbor discovery protocol (Protocol 2) on TelosB and IRIS. Dashed lines represent prediction using (2).

13. DISCUSSION

In this section, we first describe how KALwEN can be applied to User Scenarios 1 & 2 in Section 1. Then, we compare in detail KALwEN with three complete BSN key management schemes: BLIG [35,36], the KLS scheme [37] as well as the LYLR scheme [38]. Finally, we address some of the potential criticism against KALwEN.

13.1. Applying KALwEN to user scenarios 1& 2

Suppose an elderly COPD patient is prescribed a BSN. The BSN is for gait monitoring and fall prevention, and for monitoring the patient's ECG and breathing. KALwEN enables the gait monitoring sensors to communicate securely with the fall prevention actuators and *vice versa*.

When the gait monitoring sensors detect a fall, they broadcast a PANIC signal. The ECG and respiratory sensors *authenticate* the PANIC signal, and broadcast their readings. If there is an SMS-capable device in the network, it will *authenticate* the PANIC signal, the ECG and respiratory sensor readings; and send an emergency text to the patient's caregiver. All broadcasts are secured using KALwEN's *global key* and *cluster hash chains*.

To set up a telemedicine session, the patient needs to *add* an Internet-capable LPU to the network so that his/her physiological data can be transmitted to a remote specialist. To do so, supposing the system parameter $\tau = 2$, the patient bootstraps the LPU with two of already deployed sensors in the SFC. After bootstrapping, the LPU is able to join the network. After the telemedicine session, the patient simply *removes* the LPU from the network.

13.2. Comparison

A mainly qualitative comparison is given in Table III. It is impossible to compare all four protocols on an equal footing, because of the different security objectives and threat models used. Perhaps the only commonality of all proposals is that they all use public-key cryptography. BLIG is the simplest scheme because it does not support batch deployment and considers forward key secrecy a violation of functionality. KALwEN has relatively high complexity because of the harsh imposed constraints and the stringent security objectives. KALwEN caters for the absence of an LPU, and makes allowance for nodes to be removed from the user's body without the user's knowledge. The following distinction is particularly important: in other schemes, a compromised node can forge broadcast packets using the identity of other nodes; in KALwEN, this attack is not feasible because broadcast is authenticated. In the KLS scheme, a healthcare worker is absolutely trustworthy. In KALwEN, the user trusts only himself/herself. In the LYLR scheme, pairwise keying material are transmitted encrypted using the group key, allowing every node to know the pairwise key of every pair of nodes in the network—this is the price that LYLR pays for its better bootstrap performance than KALwEN. KALwEN makes the best effort in isolating nodes from each other to minimize the impact of a single node's compromise.

In terms of equipment, BLIG relies on the user to confirm every node addition by comparing blinking patterns; the KLS scheme adds on BLIG an extra layer of security—a healthcare worker; the LYLR scheme relies on the user to perform visual comparison of blinking patterns and to count the number of nodes; these are the compromises they make for dispensing with a Faraday cage. Our design is to make operation as easy and foolproof for the user as possible—no cognitive effort for comparing blinking patterns, no counting, no need to add nodes one by one—therefore the SFC is introduced.

In terms of formal proof, the node discovery protocol of the KLS scheme is proved by the Burrows-Abadi-Needham (BAN) logic [75]. The catch of BAN logic is that it assumes all principals are honest, and the KLS scheme makes exactly this assumption. Our protocols are proved in the stronger Dolev-Yao model using theorem provers.

Table III. Detailed comparison of BSN key management schemes.

BLIG [35,36]	KLS [37]	LYLR [38]	KALwEN
Devices per user			
An authorization node and regular nodes	Patient's controller, healthcare worker's controller and regular nodes	User's controller and regular nodes	User's SFC and regular nodes
Supported secure communication modes and corresponding key types			
Confidential but unauthenticated broadcast: group key	Confidential but unauthenticated broadcast: group key	• Secure broadcast by the controller: group key and hash chain • Confidential but unauthenticated broadcast by the nodes: group key • Secure unicast: pairwise key	• Secure broadcast by the KDC (delegated): global key and global hash chain • Secure broadcast by the nodes: global key and cluster hash chain • Secure unicast: pairwise key
Batch deployment			
No	No	Yes, using the group device pairing protocol: • Group key is established using a multi-party extension of Diffie-Hellman key agreement [76] • The controller distributes the necessary keying material for authenticated broadcast and secure unicast to the nodes, encrypted using the group key	Yes, using the bootstrap protocol (Protocol 1)
Performance on TelosB: N/A	N/A	25 s for 3–10 nodes	5–40 s for 2–12 nodes
Estimated complexity: N/A	N/A	H	H
Node addition			
• The joining node initiates a public-key-based protocol with the authorization node and a deployed node • The authorization node creates a log entry • The deployed node gives the joining node the group key	• The patient's controller initiates a public-key-based protocol with a newly discovered node and the healthcare worker's device • The new node creates and shares a pairwise key with the patient's controller • The health care worker's device sends a signed authorization message to the patient's controller and to the new node	A few more steps on top of the group device pairing protocol	A few more steps on top of the bootstrap protocol
Estimated complexity: H	M	H	H
Node removal			
Does not support forward key secrecy	• Intentional removal: the patient uses its controller to distribute a new group key • Malicious removal: detection mechanism is unspecified	• Intentional removal: the patient uses its controller to distribute a new group key using logical key hierarchy [77] for better efficiency than the preceding scheme • Malicious removal: detection mechanism is unspecified	• A node listens for keep-alive packets from its neighbors • Failure to receive at least $\tau - 1$ keep-alive packets within a keep-alive interval indicates removal (either intentional or malicious) and keying material is erased
Estimated complexity: N/A	L	M	L
Formal security proofs			
No	Node discovery protocol proved by BAN logic	No	Automated proof using theorem prover ProVerif and Scyther
Security level: L	L	M	H

N/A = Not Applicable, L = Low, M = Medium, H = High.

13.3. Answers to potential criticism

'Is interoperability a realistic requirement?' For the ALwEN project, Roessingh Research and Development[**] is making activity sensors whereas Holst Center[††] is making ECG sensors. These sensors *are* from different makers and they *must* interoperate with each other. Li *et al*. s work [38] also supports our interoperability requirement. On the contrary, we think it is unrealistic to assume a manufacturer would make all the sensors that end up in a BSN.

'Why not use factory pre-assigned symmetric keys?' We first preclude the possibility of assigning the same key to all devices from the same manufacturer. Suppose every device has a unique pre-assigned key, and suppose a user has n devices. The user needs to manually distribute $n - 1$ keys (of other devices) to every device, so that the devices can establish a group key, amongst other keys. This is a combined effort of $O(n^2)$ on the user's behalf. Furthermore, there is the logistics problem: the user has to keep track of the pre-assigned keys. Obviously, the keys must not be printed on the devices.

'Nodes do not need to communicate with each other.' In some applications, the nodes *do need* to interact with each other; for example, for time synchronization, context generation. See User Scenario 2 for a concrete example, which motivates the requirement for in-network processing.

'The SFC should be replaced by the personal computer.' While the personal computer (PC) has become a common household appliance, computer literacy among the elderly remains low. In addition, the SFC offers several definite advantages over the PC:

- The SFC supports batch deployment in a closed secure environment, whereas the PC does not.
- The SFC presents a dedicated interface to the user and is hassle-free in terms of software maintenance.
- The average PC nowadays is under constant threats of cyber-attacks from the Internet. The number of viruses was estimated to have topped 1 million in 2009 [78]. According to the Anti-Phishing Working Groups Phishing Activity Trends Report for Q3 of 2009, 48.35% of 22 754 847 scanned computers remain infected with malware. The problem has become so serious that in Australia several drastic measures have been recommended in the recent parliamentary report 'Hackers, Fraudsters and Botnets: Tackling the Problem of Cyber Crime'. One of the recommendations is to disconnect infected computers until they are disinfected. The standalone nature of the SFC thus compares favorably with the virus-ridden state of the PC.

'The SFC is impractical.' The introduction of the SFC can be seen as a necessary evil, because an out-of-range attacker can eavesdrop on short-range communication using only low-cost equipment [56]. We do not believe the SFC should be a usability barrier based on the reasoning that it does not yet exist, because wireless sensors did not exist until a few years ago. The bootstrap and node addition procedures consist of only four steps each. We also note that other existing designs such as Kuo *et al*.'s [11] and Blink 'Em All [22] also employ special equipment (Faraday cage and video camera, respectively) for bootstrapping multiple nodes at the same time.

While an estimation of the production cost of the SFC cannot be provided, the intention here is to provide an outline of an inexpensive construction. A basic SFC requires only the following components:

1. a dial and three light indicators;
2. a high-end sensor node as the KDC;
3. a low-end sensor node as the RF jammer;
4. interface logic connecting the dial and indicators to the KDC and the RF jammer;
5. a tamper-evident metal enclosure.

We mention in Assumption 1 that the SFC is tamper-resistant, but in practice, it is sufficient to make the SFC tamper-evident. So long as the user does not use the SFC when he/she spots evidence of tampering, there is no security breach.

'Why use the SFC as the KDC?' For one, the SFC is more resourceful than the nodes. Secondly, it is more cost-effective to make the SFC tamper-resistant than to make every node tamper-resistant. Thirdly, this is due to the relaxation of the assumption that there is an LPU.

'Is the design really suitable for the low power operation of BSNs?' Experimental results provided in Section 12 on two major sensor network platforms—TelosB and IRIS—strongly indicate that KALwEN is indeed suitable for the low operation of BSNs.

'The keys-erased-once-switched-off mechanism is impractical because node resets due to exception handling could be common.' The switching off of a node by a user is a clear sign that the user is making a conscious decision to remove the node from the network. The resetting of a node due to exception handling is a clear sign that something is wrong with the software. Proper exception handling does *not* reset a node. Unhandled exceptions such as memory access errors could reset a node. Unhandled exceptions such as infinite loops do require a node to be reset, but if a healthcare device frequently encounters such errors, it is unfit for healthcare purposes in the first place.

14. 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 com-

[**] http://www.rrd.nl

[††] http://www.holstcentre.com

plete 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. The cryptographic primitives used by KALwEN are lightweight enough to run on sensor node hardware. KALwEN is decentralized so that it does not depend on the availability of an LPU. While supporting secure global broadcast, local broadcast, and local unicast, KALwEN is able to preserve past key secrecy and FKS. The fact that all the cryptographic protocols of KALwEN have been formally verified also makes a convincing case. Experimental results showing that bootstrapping 12 IRIS nodes takes a little under 25 s are a strong indication that KALwEN is practical. KALwEN at this stage is a proposal for addressing the constraints in Section 1; more tests are pending for substantiating the usability claim.

Appendix: ProVerif script for Protocol 1

```
(* A public channel. *)
free net.

(* A private channel associated with
 the SFC.*)
private free sfc.

(* Message tags. *)
free hello,boot.

(* The name of a compromised agent. *)
free spy.

(* Private channels for agent
 initalization. *)
private free initialInitiatorData.
private free initialResponderData.

(* Symmetric encryption functions. *)
fun encrypt/2.
reduc decrypt(encrypt(x,y),y)=x.
 (* y = the key *)
fun mac/2.
fun h/1.

(* Cryptographic constructors. *)
fun pencrypt/2.
            (* asymmetric encryption *)
fun enc/1.
          (* extracts encryption key *)
fun dec/1.
          (* extracts decryption key *)

(* Cryptographic destructors. *)
reduc pdecrypt(pencrypt(x,enc(y)),
 dec(y)) = x.

(* Constructor maps agents to secret
 key-pairs. *)
private fun keypair/1.

(* A lookup function for public keys. *)
reduc pubkey(agent)
                = enc(keypair(agent)).

(* Diffie-Hellman (DH) function *)
fun dh/2.
fun g/1.
equation dh(x,g(y)) = dh(y,g(x)).

(* The queries. *)
private free secretI, secretR.
query
  attacker:secretI; attacker:secretR;
  evinj:endInitAuth(x,y)
    ==>evinj:beginInitAuth(x,y);
  evinj:endRespAuth(x,y)
    ==>evinj:beginRespAuth(x,y).

(* The initiator process. *)
let initiator =
  in(initialInitiatorData, (idv,rv));
  ! (* model arbitrary no. of
    sessions *)
  new nv;
  out(sfc, (idv, hello, g(rv), nv));
  in(sfc, (msgv, macv));
  let (ids, =idv, =boot, grs, ctextv,
      ns) = msgv in
  let ksv = dh(rv, grs) in
  let km = decrypt(ctextv, ksv) in
  if mac((nv, msgv), ksv) = macv then
    (if ids<>spy then event endRespAuth
        (ids, idv))
       | event beginInitAuth(idv, ids);
    out(sfc, mac(ns, ksv));
  out(net, encrypt(secretI, ksv));
  out(net, encrypt(secretI, km));
  0.

(* The responder process. *)
let responder =
  in(initialResponderData, (ids,rs,km));
  !(* model arbitrary no. of sessions *)
  in(sfc, (idv, =hello, grv, nv));
  new ns;
  let ksv = dh(rs, grv) in
  let ctextv = encrypt(km, ksv) in
  let msgv = (ids, idv, boot, g(rs),
           ctextv, ns) in
  let macv = mac((nv, msgv), ksv) in
  event beginRespAuth(ids, idv);
  out(sfc, (msgv, macv));
  in(sfc, =mac(ns, ksv));
  if idv<>spy then event endInitAuth
          (idv, ids);
  out(net, encrypt(secretR, ksv));
  out(net, encrypt(secretR, km));
  0.

(* The initializer process. *)
let initializer =
  new agent; (* generate agent name *)
  new km;    (* membership key *)

  (* compute public encryption
     key-pair *)
  let pkp = keypair(agent) in

  (* launch initiator role *)
  out(initialInitiatorData,
     (agent,dec(pkp)));

  (* launch responder role *)
  out(initialResponderData,
     (agent,dec(pkp),km));

  (* publish the public data *)
  out(net, (agent, enc(pkp))).

(* The compromised agent. *)
let compromised =
  (* compute public encryption
     key-pair *)
  let pkp = keypair(spy) in

  (* publish the key-pair to model
     compromise *)
  out(net, pkp).

(* The system. *)
process
    !initiator  (* initiators *)
  | !responder  (* responders *)
  | compromised (* the spy *)
  | !initializer
```

ACKNOWLEDGEMENT

Yee Wei Law and Marimuthu Palaniswami are supported by the Australian Research Council Research Network on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), and the ARC grant DP1095452.

Zheng Gong and Pieter Hartel acknowledge the financial support of SenterNovem for the ALwEN project, grant PNE07007.

The authors thank Hermie Hermens for his input on the use cases; and Bo Zhu and Jacob Andersen for their code and discussion.

REFERENCES

1. Yang GZ (ed.). *Body Sensor Networks*. Springer-Verlag: London, 2006.
2. Lo B, Yang G. Key technical challenges and current implementations of body sensor networks. *Proceedings of the Second International Workshop on Wearable and Implantable Body Sensor Networks*, 2005; 1–5.
3. Anderson R. A security policy model for clinical information systems. *IEEE Symposium on Security and Privacy*, IEEE Computer Society: Los Alamitos, CA, USA, 1996; 30–43.
4. Halperin D, Heydt-Benjamin TS, Ransford B, *et al*. Pacemakers and implantable cardiac defibrillators: software radio attacks and Zero-Power defenses. *29th IEEE Symposium on Security and Privacy*, IEEE Computer Society: Oakland, California, 2008; 129–142.
5. Kannus P, Khan KM, Lord SR. Editorials: preventing falls among elderly people in the hospital environment. *The Medical Journal of Australia* 2006; **184**(8): 372–373.
6. Yoshida M, Kan D, Takeoka M, Mouri S. Fall prevention by vibration stimuli to planta pedis. *4th European Conference of the International Federation for Medical and Biological Engineering*, Springer, 2009; 2124–2127.
7. Perrig A, Szewczyk R, Wen V, Culler D, Tygar J. SPINS: Security Protocols for Sensor Networks. *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, ACM Press, 2001; 189–199.
8. Anderson R, Chan H, Perrig A. Key infection: smart trust for smart dust. *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP)*, 2004; 206–215.
9. Zhu S, Setia S, Jajodia S. LEAP+: Efficient security mechanisms for large-scale distributed sensor networks. *ACM Transactions on Sensor Networks* 2006; **2**(4): 500–528.
10. Deng J, Hartung C, Han R, Mishra S. A practical study of transitory master key establishment for wireless sensor networks. *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, 2005; 289–302.
11. Kuo C, Luk M, Negi R, Perrig A. Message-in-a-bottle: user-friendly and secure key deployment for sensor nodes. *SenSys '07: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, ACM, 2007; 233–246.
12. Seshadri A, Luk M, Perrig A, van Doorn L, Khosla P. SCUBA: Secure Code Update By Attestation in sensor

networks. *WiSe '06: Proceedings of the 5th ACM Workshop on Wireless Security*, ACM, 2006; 85–94.

13. Seshadri A, Luk M, Perrig A. SAKE: Software attestation for key establishment in sensor networks. *Distributed Computing in Sensor Systems*, *LNCS*, Vol. 5067. Springer Berlin: Heidelberg, 2008; 372–285.
14. Castelluccia C, Francillon A, Perito D, Soriente C. On the difficulty of software-based attestation of embedded devices. *ACM Conference on Computer and Communications Security*, 2009; 400–409.
15. Eschenauer L, Gligor V. A key-management scheme for distributed sensor networks. *Proceedings of 9th ACM Conference on Computer and Communications Security*, ACM Press, 2002; 41–47.
16. Liu D, Ning P, Li R. Establishing pairwise keys in distributed sensor networks. *ACM Transactions on Information and System Security* 2005; **8**(1): 41–77.
17. Du W, Deng J, Han YS, Varshney PK, Katz J, Khalili A. A pairwise key predistribution scheme for wireless sensor networks. *ACM Transactions on Information and System Security* 2005; **8**(2): 228–258.
18. Lee J, Stinson D. A combinatorial approach to key predistribution for distributed sensor networks. *IEEE Wireless Communications and Networking Conference* 2005; **2**: 1200–1205, DOI: 10.1109/WCNC.2005.1424679
19. Chakrabarti D, Maitra S, Roy B. A key pre-distribution scheme for wireless sensor networks: merging blocks in combinatorial design. *International Journal of Information Security* 2006; **5**(2): 105–114.
20. Çamtepe S, Yener B. Combinatorial design of key distribution mechanisms for wireless sensor networks. *IEEE/ACM Transactions on Networking* 2007; **15**(2): 346–358.
21. Hoepman JH. Ephemeral pairing on anonymous networks. *Security in Pervasive Computing*, *LNCS*, Vol. 3450. Springer Berlin: Heidelberg, 2005; 101–116.
22. Saxena N, Uddin MB. Blink 'Em All: scalable, user-friendly and secure initialization of wireless sensor nodes. *CANS '09: Proceedings of the 8th International Conference on Cryptology and Network Security*, Springer-Verlag: Berlin, Heidelberg, 2009; 154–173.
23. Laur S, Nyberg K. Efficient mutual data authentication using manually authenticated strings. *Cryptology and Network Security*, *LNCS*, Vol. 4301. Springer Berlin: Heidelberg, 2006; 90–107.
24. Alpern B, Schneider FB. Key exchange using ' keyless cryptography'. *Information Processing Letters* 1983; **16**(2): 79–81.
25. Castelluccia C, Mutaf P. Shake them up!: a movement-based pairing protocol for CPU-constrained devices. *MobiSys '05: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, ACM, 2005; 51–64.
26. Varshavsky A, Scannell A, LaMarca A, de Lara E. Amigo: proximity-based authentication of mobile devices. *UbiComp 2007: Ubiquitous Computing*, *LNCS*, Vol. 4717. Springer Berlin: Heidelberg, 2007; 253–270.
27. Mayrhofer R, Gellersen H. Shake well before use: authentication based on accelerometer data. *Pervasive Computing*, *LNCS*, Vol. 4480. Springer Berlin: Heidelberg, 2007; 144–161.
28. McCune JM, Perrig A, Reiter MK. Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication. *IEEE Symposium on Security and Privacy*, Vol. 0, IEEE Computer Society: Los Alamitos, CA, USA, 2005; 110–124.
29. Buhan I, Boom B, Doumen J, Hartel P, Veldhuis R. Secure *ad-hoc* pairing with biometrics: SAfE. *International Journal of Security and Networks Special (IJSN) Special Issue on Secure Spontaneous Interaction* 2009; **4**(1): 27–42.
30. Cherukuri S, Venkatasubramanian K, Gupta S. BioSec: a biometric based approach for securing communication in wireless networks of biosensors implanted in the human body. *Proceedings of International Conference on Parallel Processing Workshops*, 2003; 432–439.
31. Bao SD, Zhang YT, Shen LF. Physiological signal based entity authentication for body area sensor networks and mobile healthcare systems. *27th Annual International Conference of the Engineering in Medicine and Biology Society (IEEE-EMBS 2005)*, 2005; 2455–2458.
32. Poon C, Zhang YT, Bao SD. A novel biometrics method to secure wireless body area sensor networks for telemedicine and m-health. *IEEE Communications Magazine* 2006; **44**(4): 73–81.
33. Venkatasubramanian KK, Banerjee A, Gupta SKS. EKG-based key agreement in Body Sensor Networks. *IEEE Conference on Computer Communications Workshops (INFOCOM)*, IEEE, 2008; 1–6.
34. Sufi F, Khalil I, Khalil I. Polynomial distance measurement for ECG based biometric authentication. *Security and Communication Networks* 2010; **3**(4): 303–319.
35. Andersen J, Bardram JE. BLIG: A New Approach for Sensor Identification, Grouping, and Authorisation in Body Sensor Networks. *Proceedings of the 4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007)*, *IFMBE Proceedings*, Vol. 13. Springer Verlag, 2007; 223–229.
36. Andersen J. Secure group formation protocol for a medical sensor network prototype. *Fifth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2009)*, IEEE, 2009; 343–348.
37. Keoh S, Lupu E, Sloman M. Securing body sensor networks: sensor association and key management. *IEEE*

International Conference on Pervasive Computing and Communications (PerCom 2009), 2009; 1–6.

38. Li M, Yu S, Lou W, Ren K. Group device pairing based secure sensor association and key management for body area networks. *INFOCOM 2010*; 1–9, March 2010.
39. Malasri K, Wang L. Addressing security in medical sensor networks. *HealthNet '07: Proceedings of the 1st ACM SIGMOBILE International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments*, ACM, 2007; 7–12.
40. Malasri K, Wang L. Design and implementation of a secure wireless mote-based medical sensor network. *UbiComp '08: Proceedings of the 10th International Conference on Ubiquitous Computing*, ACM: New York, NY, USA, 2008; 172–181.
41. Mišić J, Mišić V. Enforcing patient privacy in healthcare WSNs through key distribution algorithms. *Security and Communication Networks* 2008; **1**(5): 417–429.
42. Diffie W, Hellman M. New directions in cryptography. *IEEE Transactions on Information Theory* 1976; **IT-22**(6): 644–654.
43. Certicom Research. *Standards for Efficient Cryptography. SEC 1: Elliptic Curve Cryptography* (1st edn). September 2000.
44. Liu A, Ning P. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. *7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*, IEEE Computer Society, 2008; 245–256.
45. Law L, Menezes A, Qu M, Solinas J, Vanstone S. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography* 2003; **28**: 119–134.
46. Moharrum M, Eltoweissy M, Mukkamala R. Dynamic combinatorial key management scheme for sensor networks. *Wireless Communications and Mobile Computing* 2006; **6**(7): 1017–1035.
47. Sánchez D, Baldus H. A deterministic pairwise key pre-distribution scheme for mobile sensor networks. *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, 2005; 277–288.
48. Morchón OG, Baldus H, Sánchez DS. Resource-efficient security for medical body sensor networks. *International Workshop on Wearable and Implantable Body Sensor Networks*, IEEE Computer Society: Los Alamitos, CA, USA, 2006; 80–83.
49. Lamport L. Password authentication with insecure communication. *Communications of the ACM* 1981; **24**(11): 770–772, DOI: http://doi.acm.org/10.1145/358790.358797
50. Bradford PG, Gavrylyako OV. Foundations of security for hash chains in *ad hoc* networks. *Cluster Computing* 2005; **8**(2): 189–195.
51. Perrig A. The BiBa one-time signature and broadcast authentication protocol. *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, ACM: New York, NY, USA, 2001; 28–37.
52. Bradford PG, Gavrylyako OV. Hash chains with diminishing ranges for sensors. *International Journal of High Performance Computing and Networking* 2006; **4**(1/2): 31–38.
53. Rogaway P, Shrimpton T. Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. *Fast Software Encryption*, *LNCS*, Vol. 3017. Springer-Verlag, 2004; 371–388.
54. Coron JS, Dodis Y, Malinaud C, Puniya P. Merkle-Damgård Revisited: How to construct a hash function. *Advances in Cryptology—CRYPTO 2005, 25th Annual International Cryptology Conference*, *LNCS*, Vol. 3621. Springer-Verlag, 2005; 430–448.
55. Shamir A. How to share a secret. *Communications of the ACM* 1979; **22**: 612–613.
56. Hancke GP. Eavesdropping attacks on high-frequency RFID tokens. *Proceedings of the 4th Workshop on RFID Security (RFIDsec'08)*, 2008; 100–113.
57. Dolev D, Yao A. On the security of public key protocols. *IEEE Transactions on Information Theory* 1983; **29**(2): 198–208.
58. Hartung C, Balasalle J, Han R. Node compromise in sensor networks: the need for secure systems. *Technical Report CU-CS-990-05*, University of Colorado at Boulder, January 2005.
59. Douligeris C, Serpanos DN. *Network Security: Current Status and Future Directions*. Wiley-IEEE Press: Canada, 2007.
60. Bellare M, Rogaway P. Random oracles are practical: a paradigm for designing efficient protocols. *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications Security*, ACM, 1993; 62–73.
61. Steiner M, Tsudik G, Waidner M. CLIQUES: A new approach to group key agreement. *Proceedings of 18th International Conference on Distributed Computing Systems*, 1998; 380–387.
62. Stinson DR. Some observations on the theory of cryptographic hash functions. *Designs, Codes and Cryptography* 2006; **38**(2): 259–277.
63. Lenstra AK, Verheul ER. Selecting cryptographic key sizes. *Journal of Cryptology* 2001; **14**(4): 255–293.
64. Karlof C, Sastry N, Wagner D. TinySec: a link layer security architecture for wireless sensor networks. *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, ACM Press, 2004; 162–175.

65. Law YW, Doumen J, Hartel P. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks* 2006; **2**(1): 65–93.
66. Bellare M, Kilian J, Rogaway P. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences* 2000; **61**(3): 362–399.
67. Gong Z, Hartel P, Nikova S, Zhu B. Towards secure and practical MACs for body sensor networks. *Progress in Cryptology—INDOCRYPT 2009*, *LNCS*, Vol. 5922. Springer Berlin: Heidelberg, 2009; 182–198.
68. Healy M, Newe T, Lewis E. Efficiently securing data on a wireless sensor network. *Journal of Physics: Conference Series* **76**, 2007; DOI:10.1088/1742-6596/76/1/012063
69. Szczechowiak P, Oliveira LB, Scott M, Collier M, Dahab R. NanoECC: testing the limits of elliptic curve cryptography in sensor networks. *EWSN 2008*, *LNCS*, Vol. 4913. Springer Berlin: Heidelberg, 2008; 305–320.
70. Driessen B, Poschmann A, Paar C. Comparison of innovative signature algorithms for WSNs. *WiSec '08: Proceedings of the First ACM Conference on Wireless Network Security*, ACM: New York, NY, USA, 2008; 30–35.
71. Certicom Research. *Standards for Efficient Cryptography. SEC 2: Recommended Elliptic Curve Domain Parameters* (1st edn). September 2000.
72. Winternitz RS. A secure one-way hash function built from DES. *Proceedings IEEE Symposium on Security and Privacy*, IEEE Press, 1984; 88–90.
73. Borst J, Preneel B, Rijmen V. Cryptography on smart cards. *Computer Networks* 2001; **36**(4): 423–435.
74. Fan J, Verbauwhede I. Hyperelliptic curve processor for RFID tags. *RFIDSec09* 2009; 130–139.
75. Burrows M, Abadi M, Needham R. A logic of authentication. *ACM Transactions on Computer Systems* 1990; **8**(1): 18–36.
76. Dutta R, Barua R. Provably secure constant round contributory group key agreement in dynamic setting. *IEEE Transactions on Information Theory* 2008; **54**(5): 2007–2025.
77. Harney H, Harder E. Logical key hierarchy protocol. *Internet Draft*, IETF, March 1999.
78. Pauli D. Number of viruses to top 1 million by 2009. Available at: http://www.computerworld.com/s/article/9075118/Number_of_viruses_to_top_1_million_by_2009 April 2008.