A Shared-Secret Free Security Infrastructure for Wireless Networks

LIFENG SANG and ANISH ARORA, The Ohio State University

This article develops a shared-secret free wireless security infrastructure that provides confidentiality, identity authentication, message authentication, integrity, sender nonrepudiation, receiver nonrepudiation, and anonymity. Our infrastructure is based on two physical primitives, namely collaborative jamming and spatial signature enforcement, and a zero knowledge alternative for bootstrapping trust. Notably, it eschews the use of shared secrets, while providing a cryptosystem that is no less secure than conventional cryptosystems.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms: Algorithms, Performance, Design, Implementation

Additional Key Words and Phrases: Wireless security, sensor networks, physical primitive

ACM Reference Format:
Sang, L. and Arora, A. 2012. A shared-secret free security infrastructure for wireless networks. ACM Trans. Autonom. Adapt. Syst. 7, 2, Article 23 (July 2012), 21 pages.
DOI = 10.1145/2240166.2240173 http://doi.acm.org/10.1145/2240166.2240173

1. INTRODUCTION

Wireless security design has typically been based on cryptosystems and protocols used in wired networks. As such, the use of shared secrets as a cryptographic basis is the norm today in wireless security [Fischer and Green 2003]. Nevertheless, wireless networks differ from their wired counterparts in a number of ways: they rely on a broadcast medium, often have higher densities of deployment, and—especially in the context of sensor and mobile networks—have severe resource limitations. As a result, they face a substantially more severe problem of secret management [Perrig and Tygar 2003].

Towards reducing security management overhead, there has been academic interest in recent years in shared-secret free security in wireless communications. Building on Wyner's seminal result [Wyner 1975] on the possibility of secure communication when the eavesdropper's channel is degraded with respect to the legitimate receiver's channel, diverse analyses have been performed for the capacity of confidential unicast: some models exploit feedback on receiver/channel state [Lai et al. 2008]; [Tekin and Yener 2008], some exploit the ability of the receiver to selectively jam the transmitter at secretly chosen times, some exploit multipath [Sayeed and Perrig 2008], and others exploit power-level selection. Although the focus thus far largely been on theory, we

This article is based on the following papers: (1) "A Wireless Security Framework without Shared Secrets" in the 11th International Symposium on Stabilization, Safety and Security of Distributed Systems, 2009; (2) "A Zero Knowledge Alternative for Bootstrapping Trust" in the International Workshop on Collaborative Trusted Sensing 2009; and (3) "Designing Physical Primitive for Secure Communication In Wireless Sensor Networks", L. Sang's Ph.D. dissertation.

Authors' addresses: L. Sang (corresponding author), A. Arora, Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210; email: lfsang@gmail.com.

DOI 10.1145/2240166.2240173 http://doi.acm.org/10.1145/2240166.2240173

recently took a first step in the development of codes for confidential communications at the physical layer [Arora and Sang 2009], using the physical primitive of collaborative jamming.

In addition to this primitive for message confidentiality, recent work has also considered message authentication without the use of shared secrets [Sang and Arora 2008b]; [Danev and Capkun 2009; Patwari and Kasera 2007]. In Sang and Arora [2008b], we proposed the concept of the "spatial signature" of a node, which is a physical characterization of the signal that the node induces at each of its neighbors. We also designed a lightweight and robust primitive that validates the spatial signature of messages at runtime, and illustrated the use of the primitive for achieving message authentication without shared secrets.

Given these physical primitives for shared-secret free secure communications, we focus on the following question in this article: *"Do physical primitives suffice to develop shared-secret free protocols for conventional security properties?"* We show for the first time that a wide variety of security services are possible without any shared secrets and with only a small basis of physical primitives. Specifically, we design a wireless security infrastructure that encompasses confidentiality, authentication, integrity, non-repudiation, and anonymity, not only for single-hop and end-to-end communications, but unicast and broadcast contexts as well. In the design of the spatial signature, we assumed there is a period of trust in which nodes can safely learn neighbors' signature. In this article, we aim to close this gap by proposing a zero knowledge alternative for bootstrapping trust. In other words, the zero knowledge alternative handles the initialization problem and helps nodes to establish the trust, based on which, they can learn each other's spatial signature in a trusted way.

The implications of reducing or eliminating shared secrets for achieving secure communications are manifold: (i) the key management overhead is greatly reduced (if not eliminated) when there is no shared secret at all; (ii) shared-secret free physical layer security can be seen as a way of increasing the security level of wireless networks wherein, say for reasons of resource limitation, a highly secure protocol cannot be implemented at higher layers; (iii) it can be used as a building block in efficiently bootstrapping the security parameters and configuration data required by higher-layer protocols and applications; bootstrapping is widely regarded as being a hard and important problem for deeply embedded and potentially large-scale wireless networks; (iv) finally, it is conceivable that in some application scenarios it is possible to completely eschew the use of shared-secret-based cryptosystems.

The rest of this article is organized as follows. We review physical primitives for cooperative jamming and spatial signature enforcement in Section 2. In Section 3, we develop a rich suite of security protocols based on these physical primitives. In Section 4, we discuss how to bootstrap trust during initialization. We make concluding remarks as well as discuss future work in Section 5.

2. TWO PHYSICAL PRIMITIVES

In this section, we briefly review two physical primitives with which we will build security protocols. Table I contains the notations that we will use in the rest of the article to abbreviate these primitives and other protocol concepts.

2.1. Cooperative Jamming

In Arora and Sang [2009], we proposed a cooperative jamming primitive that enables the design of perfectly secure message communication. By perfectly secure, we mean that an eavesdropper cannot decode the message any better than it would by random guessing, even if it completely knew the coding and decoding schemes used for communication. We also introduced the following secure coding problem.

Table I. Notation

j, k, l, e	nodes in the system
$\mathcal{S}$	domain of private messages
$\mathcal{X}$	domain of messages sent by j
$\mathcal{Y}$	domain of messages received by k
$\mathcal{Z}$	domain of messages overheard by e
Γ	domain of random time sequences
$j \langle k$	j shouts out a message, received by k
$j \langle\!\sim_\gamma k$	j shouts out a message, selectively jammed by k with random jamming sequence γ
N_j	the neighboring nodes of j
T_j	the trusted base of j
$j \langle\!\| k$	j shouts out a message designated to k, authenticated by T_j via physical primitive
f, ϕ	coding functions s.t. $\phi(y, \gamma) \equiv s$ where s is a private message that j wishes to deliver, $f(s)$ sent by j is selectively jammed by k with jamming sequence γ, and y is the message k receives

Given an arbitrary message s, $s \in \mathcal{S}$, which node j wishes to send privately to node k, and an arbitrary time sequence γ, $\gamma \in \Gamma$, which node k applies to cooperatively jam while j is sending. Design coding functions f and ϕ for $j \langle\!\sim_\gamma k$: x such that

(1) $x = f(s)$, where $x \in \mathcal{X}$;
(2) $\phi(y, \gamma) = s$, where $y \in \mathcal{Y}$;
(3) $Pr(s) = Pr(s|z)$, where $z \in \mathcal{Z}$.

Note that the third requirement directly implies the following lemma.

LEMMA 2.1. *Any solution to the secure coding problem ensures perfect secrecy of message communication from j to k.*

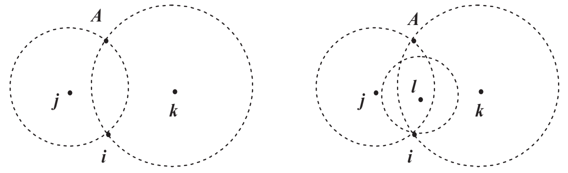
Our solution to the secure coding problem is called "dialog codes". These codes are derived from the following basic strategy: Each source bit is augmented with a redundant bit; the receiver randomly jams either bit in a pair. Since the eavesdropper does not know which bit is jammed or what the output would be, he cannot recover the jammed bit or decode the message correctly.

Let us illustrate the basic idea behind the jamming primitive in terms of the well-known "flipping model" in which a source bit gets flipped upon jamming. A simple mechanism then would be as follows. Let each bit in s be represented by two bits

$$x_{2i-1}\, x_{2i} = \begin{cases} 0\,0 & if\ s_i = 0 \\ 1\,1 & if\ s_i = 1. \end{cases} \tag{1}$$

Hence the signal transmitted from j is a stream of pairs, with values 00 or 11. k's cooperative jamming strategy is to jam either position of each pair, and to recover the input simply by looking at the remaining bit within each pair. Since the bit corruption resulting from jamming is deterministic (i.e., value flipping) as a result of the definition of the channel model, what the eavesdropper sees would always be either 01 or 10, which is equally likely the result of jamming either 11 or 00. So the probability for the eavesdropper to make a correct guess for each pair is $\frac{1}{2}$. Therefore, the eavesdropper's chance of correctly guessing s is $\frac{1}{2^m}$, where m is the number of bits in s, and that gives us perfect secrecy.

By way of an example, this time letting $s = 1101$, s would be encoded as 11 11 00 11 by j. If k were to corrupt the first bit in each pair, then the eavesdropper would receive the corrupted value 01 01 10 01 and ?1 ?1 ?0 ?1 would be received at k. k can certainly



(a) Adversary at A cannot be distinguished from i by j and k

(b) Adversary cannot imitate i without being detected by j, or k, or l

Fig. 1. Example of minimum density required for *uniqueness*.

recover s simply by looking at the second bit within each pair, however, the eavesdropper has no way of knowing whether "01" is produced by "0" or "1".

In Arora and Sang [2009], we provided an extended class of dialog codes for diverse channel models and receiver models. We also implemented a prototype and validated the algorithms on both *CC2420 (IEEE 802.15.4)* and *CC1000* platforms. The assumption of realizing this primitive is time synchronization, which is further relaxed in Section 3.3.2.

2.2. Spatial Signature Enforcement

In Sang and Arora [2008b], we proposed the concept of a node's "spatial signature", which is a physical characterization of the signal that the node induces at each of its neighbors. We showed experimentally that a spatial signature of nodes based on physical features such as Received Signal Strength Indicator (RSSI) is unique, with high probability, in multiple radio platforms and in diverse network topologies that range from rather sparse to very dense. We also designed a lightweight and robust primitive that validates and enforces the use of spatial signature in authenticating messages at runtime.

The basic idea of this primitive lies in the following proposition: if nodes are in a K-dimension space, $K + 1$ neighbors in general (i.e., nondegenerate) position suffice for defining a unique spatial signature for nodes. The geometric argument underlying this proposition is straightforward, and is in essence that of ranging-based device positioning.

We illustrate the idea with an example in $2 - D$ space. If a node i has only one neighbor j, then an adversary A located anywhere on the circle of radius $dist_{ij}$ centered at j can convince j that A is i. If i has two neighbors, j and k, as shown in Figure 1(a), A can convince both j and k that it is i by being located at point A. But in Figure 1(b), where i has three noncollinear neighbors j, k, and l, there exists no location at which A could be confused with i. By the same token, i cannot hide its identity from its signature basis (i.e., all of j, k, and l) when it sends a message. Thus, three (which is $K+1$ in this example) neighbors in a general position suffice for defining a unique spatial signature.

Based on the spatial signature primitive, we presented a simple cooperative protocol for message authentication that exploits the broadcast nature of wireless communications: any receiver in the signature basis of a node complains if a sender claims the identity of the node but the signature does not match locally. In other words, a message is authenticated if no node in the signature basis of the node complains. This idea has been validated in some other works too [Danev and Capkun 2009; Patwari and Kasera 2007]. We use $j \langle\!\| \, k$ to denote the use of spatial signatures to enforce reception of a message sent by j to k. The cooperative protocol has the following properties.

LEMMA 2.2. *Liveness: if $j \langle\!\| \, k$: x, x is delivered at k.*

LEMMA 2.3. *Safety: if x is delivered at k as a result of k's role in protocol $j \langle\!| k$: x, x must have been sent by j.*

In this protocol, we assumed in Sang and Arora [2008b] there is a period trust in which nodes can safely learn each other's spatial signature. This initialization assumption is handled by a zero knowledge alternative in Section 4.

3. SECURITY PROTOCOL SUITE BASED ON PHYSICAL PRIMITIVES

We now build upon the primitives described before to describe and validate a diverse set of security protocols, including protocols for confidentiality, identity authentication, message authentication, integrity, sender nonrepudiation, receiver nonrepudiation, and anonymity. We consider communications in a wireless network of the following sorts:

—one-hop unicast and broadcast;
—end-to-end unicast and broadcast.

3.1. System Model

The system consists of a wireless network of static, and potentially resource-constrained, nodes. Let j and k be two legitimate nodes in the system; legitimate nodes, unlike adversarial nodes, are trusted to execute their protocols correctly. And let T_j and T_k be the respective "trusted bases" of j and k; the trusted base of a legitimate node consists of at least c legitimate nodes in its one-hop neighborhood, for some strictly positive constant number c (c is typically two or three). It follows that the system has nontrivial density in the neighborhood of each node.

We assume the following system properties.

—When some node impersonates j in a communication with k, any trusted base node of j complaining of a signature failure communicates the complaint to j.
—j can discover the trusted base of k, if k is in its one-hop neighborhood.

3.2. Threat Model

The adversary in this wireless setting has, in the spirit of Dolev-Yao, the capability to: (1) eavesdrop on messages in its reception range; (2) block messages sent within its interference neighborhood; (3) replay older messages, possibly modifying the payload; and (4) inject arbitrary messages to nodes. The adversary may physically operate via devices other than the legitimate nodes or it may operate via nodes that it compromises. If a node is compromised, its state becomes known to the adversary. Compromised nodes may collude with other compromised nodes within their communication range to launch attacks.

UniSec($j \rightarrow k$): s

$j \langle\!| k$:	hello
$k \langle\!| j$:	hello_ack
$j \langle\!\sim_\gamma k$:	$f(s)$

Note that we are not in the position of dealing with communication disruption that results in more message loss, such as malicious jamming. (This problem is common to all wireless applications, and may be compensated by retransmission or routing around jammed regions.) Instead, we are interested in whether the proposed protocols are secure in the sense that the system satisfies its security properties despite the adversary, and the adversary is not able to convince the system to launch malicious or denial-of-service actions.

3.3. Confidentiality

3.3.1. One-Hop Unicast. There are numerous contexts of message communciation where one or both the sender and the receiver demand privacy: a sender communicating an "alarm" communication and a receiver receiving a "query response" may each require privacy regardless of whether the other party requires privacy or not. Now, confidentiality in this one-hop setting is readily achievable via the cooperative jamming primitive, assuming the receiver faithfully performs its cooperative role. If the receiver does not cooperate, the sender's message would be sent in the clear.

A step towards ensuring that the receiver cooperates is to coordinate with it before a transmission. Our protocol for one-hop secure unicast, denoted by $UniSec(j \rightarrow k)$, is formally stated as follows.

In this protocol, j initiates by sending k an authenticated "hello" message. After receiving "hello", k knows that j intends to send a private message. If k does not want to proceed, it can simply ignore the "hello" message; otherwise, it acknowledges with an authenticated "hello_ack" message, confirming to j that k is in sync to jam j's transmission. Note that the message authentication itself is achieved by using the spatial signature enforcement primitive for authenticating the principals.

After j receives the "hello_ack" from k, it encodes s with the coding function f, and sends $f(s)$ in the clear immediately. In sync, k jams with a random sequence γ that is (ideally) known only to itself, it can then decode the remaining information it receives to recover s via ϕ and γ, as described in the secure coding approach previously. The proof sketch for this protocol provided in Section 6.1.

While we are not able to provide details due to the space limit, one notable experimental result from our earlier work in Arora and Sang [2009] shows that the security strength for a 29-byte message (which is a typical packet size in TinyOS) of this protocol is about $\frac{1}{2^{96}}$, while the security level achieved with even a 1024-bit key using asymmetric RSA is approximately $\frac{1}{2^{80}}$. Note that the larger the message size, the stronger the security strength for our protocol. In contrast, the security strength is only dependent on key size for key-based techniques.

3.3.2. Two-Way (Two-Hop) Unicast. We develop next an extension of the previous protocol for the special case where cooperative jamming is used by two senders, as opposed to a sender-receiver pair. The extension allows two nodes to exchange information concurrently, without either communication being revealed to other nodes.

Two-Way UniSec

```
Message exchange step:
    $l \langle\!\langle j, k$: ready to help

    $j \langle l$: $s_j$
    ||
    $k \langle l$: $s_k$

    $l \langle j, k$: $s_j \boxplus s_k$
```

In Figure 3, l serves as an intermediate router for the exchange between j and k. We directly apply the cooperative jamming primitive in a symmetric form as follows: j and k transmit roughly at the same time. l shouts back the raw signal which is a combination of j's signal and k's signal. If the wireless channel follows the "addition" model, both j and k can decode the message because they know what they have sent. The router l simulates a full-duplex transceiver at both j and k; that is, if j and k were full-duplex transceivers as opposed to half-duplex, they could perform the exchange themselves

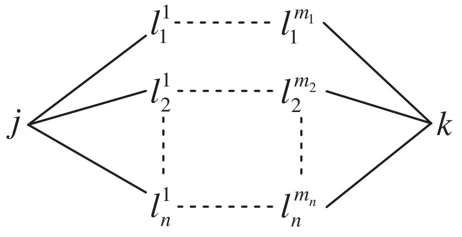


Fig. 2. End-to-end unicast confidentiality via multiple paths.

and l would not be necessary. In theory, j and k can simply subtract the signal they have sent from the signal they hear from the router, and decode the message if the channel is additive [Katti et al. 2007]. In practice, the development of coding scheme is, however, more complicated, and beyond the scope of this article. Note that this extension may by attacked by allowing l to send a bogus signal instead of what it heard. This attack is dealt with by adding a checksum/hash value at the end of the message. Since l has no way of knowing the value of s_j and s_k, it is hard for l to make up a legitimate message that passes the checksum validation.

3.3.3. End-to-End Unicast. A key requirement of confidentiality for an end-to-end unicast is that nodes in the middle of the network that forward a message should not themselves be aware of the content of the message.

In traditional shared-key security, this is easily achieved if the message is encrypted with an end-to-end shared secret (notwithstanding the difficulty of initializing the end-to-end secret). Unfortunately, with the physical primitives described earlier, it is difficult not to reveal the message to nodes in the middle. Any node that helps to hide the communication via cooperative jamming is able to decode the message if it knows the jamming time sequence. Even if nodes on the path between two parties of interest can be trusted, this approach still violates the requirement for end-to-end confidentiality.

e2eUniSec($j \rightarrow k$): s

```
$s = (s_1, s_2, \cdots, s_n)$
$\forall i, 1 \le i \le n$,
{
$UniSec(j \rightarrow l_i^1)$:                  $f(s_i)$
$UniSec(l_i^1 \rightarrow l_i^2)$:                $f(s_i)$
...
$UniSec(l_i^{m_i-1} \rightarrow l_i^{m_i})$:      $f(s_i)$
$UniSec(l_i^{m_i} \rightarrow k)$:                $f(s_i)$
}
```

A standard idea is to split the message s into multiple pieces, and to send all pieces separately along node disjoint paths. As long as intermediate nodes from all the paths do not collude, s is safe.

Our protocol for end-to-end secure unicast is formally stated in Listing 3.3.3.

A message s is first divided into n pieces assuming that there are at least n node disjoint paths between j and k, as shown in Figure 2. Let l_i^t ($t = 1, 2, \ldots, m_t$) be the nodes on path i between j and k. j first securely transmits s_i to l_i^1 via the $UniSec(j \rightarrow l_i^1)$ protocol. l_i^1 then forwards s_i to its next forwarder along the path to k securely. In the end, $l_i^{m_i}$ forwards s_i to k. The same procedure is followed on all n paths

$$\begin{tikzcd} j \arrow[r, dash] & l \arrow[r, dash] & k \end{tikzcd}$$

Fig. 3. End-to-end unicast confidentiality for line topology.

in parallel (for pedagogical reasons, we omit discussion of the scheduling necessary to deal with potential self-interference in this process). As long as the adversary does not compromise n or more nodes, with at least one each from a distinct path, the message s is safe.

3.3.4. One-Hop Broadcast. Recall that in the cooperative jamming primitive, only those nodes that know the jamming sequence can decode the message from the residual information. In essence, the jamming primitive is designed for unicast communication, as only the node that helps to jam knows the sequence that it has applied. In order to send a packet to all the neighboring nodes securely, a straightforward approach is to send the packet to each neighboring node in sequence, but the communication overhead would be high. We may optimize this approach by letting all potential receivers share a jamming sequence, which introduces a one-time cost in the initialization. All subsequent jamming sequences can be derived from the knowledge of the previous jamming sequence and the received message.

Our one-hop secure broadcast protocol is formally stated next.

A node j initializes the jamming sequence with nodes in its neighborhood, informing every node in N_j to use $\gamma_{j,0}$ as the initial jamming sequence. When a broadcast is initiated from j, it chooses one node k in N_j to help hide the first message s_1. Since k jams with $\gamma_{j,0}$, which is known to all other nodes in N_j, every node in N_j can decode $f(s_1)$ to obtain s_1, while nodes not in N_j cannot discover s_1 without the knowledge of $\gamma_{j,0}$. In all following messages, the new jamming sequence is generated by a one-way function on the previous jamming sequence. For initialization, the system can either perform this step for all the senders at the very beginning or lazily in the sense that this step can be performed only upon request. We relegate the proof to Section 6.2. Frequent update of the jamming sequence increases the level of difficulty for the adversary to break the secure broadcast.

BrdSec($j \langle N_j$): s

```
Initialize the jamming sequence (one–time cost):
    $\forall k \in N_j$ ,  $\{UniSec(j \rightarrow k):\ (\gamma_{j,0}, brd)\}$

First broadcast:
    $j \ \langle\!\sim_{\gamma_{j,1}} k:\ f(s_1)$

Second broadcast:
    $j \ \langle\!\sim_{\gamma_{j,2}} k:\ f(s_2)$
...
$m^{th}$ broadcast:
    $j \ \langle\!\sim_{\gamma_{j,m}} k:\ f(s_m)$
...
where $\gamma_{j,1} = \gamma_{j,0}$ ,  $\gamma_{j,2} = H^1(\gamma_{j,1})$ ,  and  $\gamma_{j,m} = H^{m-1}(\gamma_{j,m-1})$
```

3.3.5. Secure Flood. Flooding is a common service in many applications. We realize secure flooding by extending the *UniSec* and *BrdSec* protocols, for two cases.

Case 0: the network has no structure. If the underlying network does not provide any structure, we can simply adopt a typical flood service such as a naive flood, gossip, etc. The main difference is that the old broadcast is replaced by our *BrdSec* protocol. A duplicate suppressing mechanism may be used to reduce overhead.

Case 1: the network is partitioned into groups, each having a leader node. This approach exploits a tree structuring of the system, to reduce communication overhead. Specifically, we use a tree that spans all the group leaders, and subtrees by which each group leader spans its group. The secure flood involves using a secure one-hop broadcast within each group and a secure one-hop unicast on the leader tree. This protocol is stated as follows.

Secure flood(j)

$UniSec(j \to L_j)$:	s
$UniSec(L_j \to NL_{L_j})$:	s
$BrdSec(L_i \; \langle \; N_{L_i})$:	s

First, the source node j sends a packet securely to its group leader L_j, then L_j forwards the packet securely to its neighboring leaders NL_{L_j} and so forth. After that, each leader L_i securely broadcasts the message to all nodes in its respective group. As in other flood protocols, a duplicate suppressing mechanism may be used to minimize unnecessary retransmissions.

3.4. Authentication

3.4.1. One-Hop Unicast and Broadcast. For authentication in a one-hop setting, there is not much difference between the case of unicast and that of broadcast when using the spatial signature primitive, because a node is authenticated by another node as long as it is authenticated by all of its trusted neighbors. Identity authentication and message authentication are also quite similar in this setting.

One-hop authentication is achieved by the physical primitive alone. If s is indeed sent by j, no one in the neighborhood of j would complain by the definition of spatial signature enforcement, thus s can be authenticated. Please refer to Sang and Arora [2008b] for experimental results.

One-hop message authentication

$j \;\langle\!| \; k\colon \; s$

In identity authentication, node j claims that it is j, and this is validated by its trusted neighbors. If it is the claimed node, no one in N_j would complain, and its identity is authenticated.

Identity authentication

$j \;\langle\!| \; k\colon \; (j, s)$

3.4.2. End-to-end Unicast and Broadcast. For authentication in an end-to-end setting, similarly, there is not much difference for unicast and broadcast. To solve the end-to-end authentication problem, we assume, as is common in many network applications, that trust is transitive in the sense that if a trusts b and b trusts c, then a trusts c. The basic idea is that if in a hop-by-hop fashion each receiver trusts the preceding sender that forwards the message, then the intended destination can authenticate the message.

Letting l be a node between j and k, the protocol for end-to-end message authentication is formally stated next.

End-to-end authentication

$j \llparenthesis l: (j, s)$
$l \llparenthesis k: (l, (j, s))$

j informs l that it intends to send a message s along with its identity j. l authenticates this message and j's identity as well, and then forwards this message with its own identity (l) to k. If k authenticates l, by the transitivity property, k can authenticate that s is from j. Note that the same protocol applies to cases where there are multiple intermediate nodes along the path from j to k.

3.4.3. Authentication via Certificates. This protocol allows two nodes to mutually authenticate each other, even if they are new to each other in the system. The protocol uses a version of certificate exchange, so that the two can learn each other's signature base. Specifically, their one-hop neighbors serve as the Certification Authority (CA). As long as they have common neighbors in their trusted bases, they can enforce trust in each other through their common neighbors.

Our certificate-based authentication protocol is formally stated as follows.

Authentication via certificates

$j \llparenthesis k: T_j$
$k \llparenthesis j: T_k \cap T_j$

In this protocol, j shouts out its trusted base, T_j, which is verified by the nodes in T_j. k now knows T_j. k replies with the common nodes in both its and j's trusted bases, which in turn is verified by T_k. Although j and k do not know each other a priori, as long as $T_k \cap T_j$ is not null, they can derive trust in each other according to the transitivity of trust. A proof sketch is provided in Section 6.3.

3.5. Integrity and Nonrepudiation

For our purpose, there is not much difference between integrity, sender nonrepudiation, and authentication, especially in the one-hop case. For end-to-end integrity and sender nonrepudiation, the receiver may send back a traversal that checks whether the message has in fact been sent by the sender, by checking the sender's immediate neighbors' reception history.

Our protocol for message integrity and sender nonrepudiation is stated as follows.

One-hop sender nonrepudiation

$j \llparenthesis k: (j, s)$

k itself can verify (j, s), and j cannot lie that it has sent s because nodes in N_j have seen an authenticated message (j, s), by the definition of the $\llparenthesis$ operator.

End-to-end sender nonrepudiation

$j \llparenthesis l: (j, s)$
$l \llparenthesis k: (l, (j, s))$
$k \llparenthesis l: \texttt{verify}\ (j, s)$

In the end-to-end case, k checks back with j's immediate neighbor l to verify that j has sent s. This provides sender nonrepudiation. In both the one-hop and the end-to-end case, we may attach a fingerprint, computed by a well-known hash function, to the message, to enhance the its integrity.

For receiver nonrepudiation, we need to ensure that the receiver indeed obtains the message. We achieve this by requiring an acknowledgement, which allows the reception to be verified.

The protocol is formally stated next.

One-hop receiver nonrepudiation

```
$j \;\langle\; k:\; s$
$k \;\langle\!|\; j:\; ACK(s)$
```

Note that here we only need receiver nonrepudiation, therefore, we do not need to authenticate the message sent by j. To check that the receiver k has in fact received the message, j establishes that it has one or more ACKs for the query in its history. Note that $ACK(s)$ are authenticated by T_k so k cannot lie about receiving s. If k chooses to not respond to s, in other words, k does not reply with an ACK message after he receives s, then k violates the protocol, and be regarded as suspect.

End-to-end receiver nonrepudiation

```
$j \;\langle\; l:\; s$
$l \;\langle\; k:\; s$
$k \;\langle\!|\; l:\; ACK(s)$
$j \;\langle\!|\; l:\; \texttt{verify}\; ACK(s)$
```

For the end-to-end case, the source can query the receiver's immediate neighbors to see whether they have seen the ACK. This makes sure that k cannot lie that it has received s. The protocol fails iff all the nodes in $T_k \cap T_j$ are compromised, but in this event no other protocol can do better. If there are multiple intermediate nodes on the path between j and k, the protocol remains essentially the same, except that we need to add one or more intermediate forwarders for the message s and to verify that one or more ACKs were sent.

3.6. Anonymity

In addition to confidentiality and authenticity, anonymity—the requirement of preventing the sender's identity from being revealed—is an important one. A potential problem in realizing anonymity using physical primitives is that two or more receivers have a good chance of identifying the sender, if the sender sends a message in the clear, the receivers can learn the spatial signature of the sender and collude in the identification. The induced signature at its neighbors is likely to match with the one learned before, especially if the sender uses the same transmission power for each communication.

One simple idea then is to let a node vary its transmission power, to intentionally change its induced signature, and make neighbor collusion somewhat harder. Another idea is to let a trusted node to first help to hide the communication and to then forward the message. Because the message is jammed by the trusted node, others cannot learn the spatial signature of the sender. The trusted node that helps jam may be able to learn the signature of the message, but cannot figure out the identity of the sender by itself.

The basic procedure is as follows: First, a node that is willing to help shouts out and it is verified by its trusted base including the potential sender. Then, the sender desiring anonymity sends its message, in the presence of cooperative jamming by the helper. Finally, the helper forwards the message in the clear. Since wireless networks, especially sensor networks, enjoy node density, such a helper is expected to be easy to find.

The protocol is formally stated as follows.

3.6.1. Unicast for Anonymity

One-hop anonymity for unicast

$$\begin{array}{lll} l \;\langle\!\mid\; j: & \text{ready to help} \\ j \;\langle\!\sim_\gamma\; l: & f(s) \\ l \;\langle\; k: & s \end{array}$$

where l is a neighbor of both j and k

End-to-end anonymity for unicast

$$\begin{array}{lll} l \;\langle\!\mid\; j: & \text{ready to help} \\ j \;\langle\!\sim_\gamma\; l: & f(s) \\ l \;\langle\; h: & s \\ h \;\langle\; k: & s \end{array}$$

where h is a common neighbor of j, l and k

3.6.2. Broadcast for Anonymity

One-hop anonymity for broadcast

$$\begin{array}{lll} k \;\langle\!\mid\; j: & \text{ready to help} \\ j \;\langle\!\sim_\gamma\; k: & f(s) \\ k \;\langle\; N_j: & s \end{array}$$

End-to-end anonymity for broadcast

$$\begin{array}{lll} l \;\langle\!\mid\; j: & \text{ready to help} \\ j \;\langle\!\sim_\gamma\; l: & f(s) \\ l \;\langle\; N_l: & s \\ N_l \;\langle\; N_{N_l}: & s \end{array}$$

It is worth noting that a sender can simply shout out a message without its identity attached, so as to achieve anonymity. This is because it could lie that the message is owned by someone other than itself. However, one potential problem is that if the channel is not busy shortly before this transmission, colluding neighbors may still be able to identify that this message belongs to the sender. That is why a volunteer is exploited to help hide the identity of the sender. [1] We provide a proof in Section 6.4.

4. BOOTSTRAPPING TRUST

Bootstrapping is a hard but inevitable task in many network applications. The primary techniques so far mainly rely on message authentication, which in general has been explored extensively in the cryptographic community [Fischer and Green 2003; Rivest et al. 1978] [Fiat and Shamir 1986]; [Shamir 1984]; [Perrig and Tygar 2003]. In this

[1]In order to cover N_j, the helper may increase its transmission power to forward the message. An alternative would be to perform 2-hop anonymity forwarding with a cost of increased communication overhead.

section, we first briefly discuss the requirements and the deployment model, then we state the initialization problem followed by variants of typical authentication methods using symmetric and asymmetric secrets. In the end, we introduce our zero knowledge alternative for lightweight and efficient bootstrapping.

4.1. Bootstrapping Requirements

The main assumptions in realizing our physical primitives include the following.

(1) Loose time synchronization: in dialog codes, we require the existence of loose time sync to introduce intentional interference. This can be easily realized with current techniques, as we implemented in Arora and Sang [2009].
(2) Interference detection: in spatial signature, we assume that a node is able to detect interference (or malicious jamming) during the learning phase. This requirement is to ensure the the collection of a neighbor's spatial signature is not interrupted by malicious interference. This assumption is verified by our detection algorithm presented in Sang and Arora [2008a].
(3) Neighborhood identification: in spatial signature, when nodes start learning a neighbor's signature, it should be assured that the identity of the neighbor is indeed the claimed identity. Otherwise, the association of the signature and its corresponding identity will fail, which leads to successful impersonation. In Sang and Arora [2008b], we simply assume an initial period of trust in which nodes can safely learn neighbors' signatures. In this section, we introduce a zero knowledge alternative that aims to close this gap.

4.2. Deployment Model

We consider the following deployment model.

—multiple users and applications coexist;
—application is not deployed in the factory;
—entire set of sensor nodes need not be deployed simultaneously;
—application security is configured in the field;
—nodes may not trust each other in the network; and
—applications do not trust the media, or even the network services.

Essentially, nodes can be deployed anywhere without any prior knowledge. As a consequence, a node may not know who its neighbors would be at the beginning of when an application starts. The hardware is resource constrained. For example, Mica2, a popular platform in the sensor network community, has only a total of 4k RAM. Therefore, the design of any security infrastructure must be lightweight.

4.3. Problem Statement

The task of bootstrapping trust in such a deployment is not easy for a variety of reasons. For example, users do not trust other coexisting applications; they may even not trust the network itself; the security of an application is configured in the field; neighbors are not determined until an application is deployed on certain pieces; there is no special hardware such as TPM (trusted platform module) [Parno 2008] existing on each device. Also, considering the limitation of sensor nodes, the design of security blocks poses the following three essential requirements:

—no memory of neighboring certificates, that is, a node should be able to authenticate legal nodes without trust of certificate at the first place;
—no centralized verification, that is, there is no trusted end-to-end mechanism assumed; and

—the impact of compromised nodes in a region is contained to only their communication area. It should not have the threshold behavior in the sense that the network is broken as the number of compromised nodes exceeds a threshold.

4.4. Related Work On Bootstrapping

We summarize variants of typical authentication methods using symmetric and asymmetric secrets in the following.

4.4.1. Bootstrapping Based on Symmetric Secrets. Techniques based on symmetric secrets are usually more efficient in terms of computation, thus they have received extensive attention for the bootstrapping problem.

—*Point-to-point secret*. The most intuitive approach might be to install all the secrets before deployment, so that point-to-point authentication is achievable. For any pair of node j and k, they can mutually authenticate each other based on the shared secret $K_{j,k}$. It is simple and efficient, but: (i) has extremely high memory requirement, especially in a large deployment; (ii) is easy to break as compromising one node jeopardizes the security of the whole network.
—*Network-wide secret*. Another extreme is to let the whole network share one secret before deployment. For example, the whole network shares $K_{network}$. j and k can then bootstrap their trust based on the knowledge of $K_{network}$. It is simple and memory efficient, but easy to break. It becomes even more fragile as the entire set of nodes need not be deployed simultaneously, and the application may be configured only on a subset of nodes.
—*Group-based secret*. In this case, the memory requirement is reduced by constructing logical structure in the network. For instance, a network is first divided into groups. Nodes can then authenticate each other based on shared intragroup and intergroup secret. Another example is to form a grid such that a node can authenticate another by using the row secret and column secret. All these variants provide reasonable resilience between memory consumption and security strength, however, extra effort is required to establish and maintain the group information. When the application is only deployed on a dynamic subset of the network, the task of constructing groups becomes even harder.

4.4.2. Bootstrapping Based on Asymmetric Secrets. Bootstrapping with asymmetric secrets is quite popular for establishing trust because of the high level of security since the contaminant is limited to the compromised node(s); yet (ii) memory efficiency in the sense that it does not store all individual secrets. Common techniques include the following.

—*KDC-based infrastructure*. In this case, there is a trusted third party that allows central verification of identity. For example, if j and k want to establish trust with each other, they communicate with the KDC which helps to verify the identity following one protocol. The assumption of the existence of KDC and the ability of communicating with it from any node is, however, not easy for certain applications, especially in a large wireless deployment where end nodes may be multiple hops away from the KDC. The cost of communication also brings high overhead including energy consumption and delay.
—*Digital certificate*. A trusted authority issues certificates with his signature, and individual node performs verification by checking the authenticity of the signature on the certificate. For example, if node j wants to authenticate itself to node k, j presents its certificate to k. k can then check the digital signature to make sure j's certificate is valid. k may further challenge j to make sure j is indeed the identity that corresponds to the presented certificate. The main disadvantage of this approach is its

inefficiency and high memory requirement, especially on low-power wireless devices. Also, the chance of global secrets being revealed increases as more information about the secret is exposed.

—*Trusted Platform Module (TPM)*. Bootstrapping trust via TPM is possible, since each TPM chip has a unique and secret RSA key burned in as it is produced. The RSA key allows a device to perform platform authentication. This approach is in theory quite similar to the digital certificate approach, it is, however, more secure because the TPM limits the use of cryptographic keys and provides remote attestation and sealed storage. The main disadvantage for the use of TPM is its high cost. For a large deployment of low-power wireless networks, it is too expensive to equip a TPM on each individual device, not mentioning the operation cost.

—*Zero knowledge proof*. Traditional zero knowledge proof provides a nice property of identity authentication. Some require multiple rounds of verification (e.g., Shamir-Fiege-Fiat), while others are based on computationally hard problems (e.g., Schnorr). Although these approaches alone are not sufficient for the bootstrapping problem as they do not guarantee the validness of the certificate, they inspired researchers to explore more secure and efficient methods for bootstrapping. We relegate the discussion of our zero knowledge alternative to the next section.

4.5. A Zero Knowledge Alternative for Bootstrapping Trust

Our new zero knowledge problem is formalized as follows.

Let p be an application secret only known to the gateway, and s_j be j's private key computed by the gateway. The problem is to design function F, G, and H for the following protocol:

$j \rightarrow k$: j, $F_p(j, s_j)$
$k \rightarrow j$: n
$j \rightarrow k$: $G_p(n, s_j)$

such that: (1) k can verify j's identity via H_p; and (2) no one can impersonate any ID without knowing p, even if some nodes are compromised. Here F, G, and H are well known; F is a certificate, G is a proof, and H is a verification procedure. F, G, and H should have the following properties.

(1) F should be a one-way function in the sense that s_j should be hard to break given j and $F(j, s_j)$. By the same token, G should be a one-way function too.
(2) $F(j, s_j) \neq F(j, s'_j)$, that is, different private keys result in different certificates on the same node. This prevents adversaries from using an old compromised secret to attack new applications.
(3) $F(j, s_j) \neq F(k, s_j)$, that is, j should not be able to generate k's certificate using s_j, otherwise, j can impersonate k.
(4) It is hard to generate a valid pair s_l and $F(l, s_l)$ without the knowledge of p. Otherwise, the adversary can impersonate l (which may not exist in the system) and break the system.
(5) $G(n, s_j) \neq G(n', s_j)$, that is, a node should generate different proofs given different challenges. This prevents adversaries from using an old proof to impersonate legal identities.

Our solution exploits Shamir's identity-based signature [Shamir 1984], where the security depends on the hardness of extraction of modular roots. We now discuss what the functions F, G, H are and how to compute the secret key and certificate.

Let w be a product of two large primes, and e be a large prime which is relatively prime to $\varphi(w)$. We will use the same w and e for all the nodes. Our solution is as follows.

—s_j: (u_j, r_j). j's private secret s_j is a pair, such that $u_j^e = j\ (mod\ w)$, and r_j is simply a random number which is application specific. This u_j can be easily computed by the gateway, and it is hard to break because extraction e-th roots mod w is believed to be an exceedingly difficult computational task. s_j is provided by the gateway and only known to j.

—$F(j, s_j)$: (c_j, g_j). j's certificate has two parts: a personal certificate c_j and a group certificate g_j, such that $c_j = r_j^e\ (mod\ w)$ and $g_j = p \cdot r_j^{\theta(c_j,z)}\ (mod\ w)$. Both c_j and g_j are provided by the gateway. Note that c_j can be also computed at node j because j knows r_j, but g_j can only be computed at the gateway.

—$G(n, s_j)$: $u_j \cdot r_j^{\theta(c_j,n)}\ (mod\ w)$. The proof G value is computed at node j using u_j and r_j.

—H: v_1 and v_2. H is a logical AND of two boolean checking functions

(1) $v1$: $G(n, s_j)^e \equiv j \cdot c_j^{\theta(c_j,n)}\ (mod\ w)$, because $G(n, s_j)^e \equiv u_j^e \cdot r_j^{e\cdot\theta(c_j,n)} \equiv j \cdot c_j^{\theta(c_j,n)} (mod\ w)$.

(2) $v2$: $g_j^e \equiv b \cdot c_j^{\theta(c_j,z)}$ where $b = p^e\ (mod\ w)$ provided by the gateway, because $g_j^e \equiv p^e \cdot r_j^{e\cdot\theta(c_j,z)} \equiv b \cdot c_j^{\theta(c_j,z)}\ (mod\ w)$.

The security of this scheme depends on the inability of the cryptoanalyst to isolate u from a large number of valid signatures and p from a limited number of group certificates. The proof of soundness and completeness for this protocol is provided in the Appendix.

We summarize the difference in assumptions between our zero knowledge work and the classical approach (e.g., Schnorr or Shamir-Fiege-Fiat) as follows: classically, ZK protocols assume that the participants are able to present certificates signed by a CA. Thus a global secret key is used by the CA for each potential node in the space and then temporally reused whenever the certificates need to be changed. In contrast, we do not assume signing using a global (both spatially and temporally) key. In our work, the global public includes w and e; both e and w are used liberally but only with application-specific material, with exception of using them with u_j (which is related to node identity and is used in a classically zero knowledge way, so even the receiver does not know u_j). The global secret includes: (i) primes used to generate w; (ii) u_j, which is used in a classically zero knowledge way.

This difference of assumptions has two implications.

(1) The absence in usage of global secret (and temporal reduction in use of all locally assigned secrets, for example, application keys and application-specific node random variables) matters because the chance of global secrets being revealed increases as more information about the secret is exposed.
(2) By not needing globally signed certificates, we can reduces the need for memory on mote devices to store such certificates.

4.6. Example of Bootstrapping the Entire Network

Now we discuss how an entire network would be bootstrapped with an example in the following. At the time of deployment, the material for the zero knowledge proof is installed on every node by the base station, or coupled with an application at the time of reprogramming (as in the Kansei testbed [Kanei Testbed 2012]). Given the ability of bootstrapping trust, nodes may start introducing themselves in an authenticated way layers by layers (or hops by hops), largely dependent on the application designer. If the network has a notion of group, the bootstrapping may start from group leaders, and then bootstrap within each group as well. After the identification is ready, nodes can then exploit the physical primitives to extend security during the entire life of the application. For example, they can exchange a secret s via either the Dialog Codes or a typical cryptography method. This s can be used to provide a footprint on random messages for physical signature initialization. An illustration of using the Dialog Codes for this procedure is provided in Figure 4 in the Appendix. We use $\rhd_{zero}$ to represent the zero knowledge proof block. $j \rhd_{zero} k$ means that k is identified by j via the zero knowledge proof block.

5. DISCUSSION AND CONCLUSION

In this article, we developed a shared-secret free wireless security infrastructure based on two physical primitives: cooperative jamming and spatial signature enforcement. The former is essentially for confidential wireless communication, while the latter essentially for message authenticity. The derived protocols include a variety of common security services, including confidentiality, identity authentication, message authentication, integrity, sender nonrepudiation, receiver nonrepudiation, and anonymity, for a variety of wireless network communication contexts. We also introduced a zero knowledge alternative to bootstrap trust among nodes in a potentially large network. This work, together with the previous work on physical primitives, illustrates a new approach in the field of wireless security: that communications without shared secrets while providing the same level of security is feasible and, in fact, relatively simple.

Although we have not discussed the energy efficiency in this article, it is possible to implement confidentiality with low power not only because the primitive reduces the key management and maintenance overhead, but also because it allows concurrent exchange of secret information at the physical layer without involving complex computation. In future work, it would be interesting to deliver a low-power infrastructure that could be useful for wireless networks such as WSNs.

Since the proposed protocols rely heavily on the correctness and efficiency of the two primitives, future work includes further justification of these existing primitives and development of new alternatives to enrich the primitive set. Another direction is to provide an automatic "InstallShield" for easy deployment of the whole protocol suite. Our work on zero knowledge proof is based on a computationally hard problem. Whether there is a physical version of zero knowledge is worth investigation. Last but not least, investigating the feasibility of combining this framework with the existing cryptography-based security infrastructure to provide better domain-specific security services is worthy of further study.

6. APPENDIXES

Recall that j and k are legitimate nodes.

6.1. A Proof for UniSec($j \rightarrow k$): s

THEOREM 6.1. *$\{UniSec(j \rightarrow k)$: $s\}$ satisfies the confidentiality property: j delivers s to k without revealing the content of s.*

PROOF. In the first step, according to Lemma 2.2, j's "hello" message will arrive at k in an authenticated way. According to Lemma 2.3, k knows this "hello" must be from j. By the same token, after the second step, j knows that k agrees to cooperate, to help hide $f(s)$. This guarantees that s would not be sent in the clear. Together with Lemma 2.1, $\{UniSec(j \rightarrow k)$: $s\}$ satisfies the confidentiality property.

We now discuss why the protocol is sound in the presence of the following threats.

(1) Eavesdrop: since the first two messages are simply control messages, it is ok for the adversary to overhear the content of these two messages. According to Lemma 2.1, the third message is protected by cooperative jamming, and the adversary can not figure out s even if it overhears $f(s)$.

(2) Block: the protocol will not proceed if the adversary blocks any of the three messages. This problem also occurs in cryptography-based protocols. Howevers, s is not revealed in the presence of message blocking.
(3) Replay: according to Lemma 2.3, any replay initiated by a third node will not succeed because its identity has to be authenticated. Therefore, s is secure in the presence of replay.
(4) Injection: message injection in general could either disrupt the normal communication or fool others into receiving bogus messages. As we mentioned in Section 3.2, if the adversary aims to inject noise to introduce more loss, common techniques (e.g., retransmissions) may be used. If the adversary tries to inject bogus messages to fool either j or k, he would be caught according to Lemma 2.3. It is worth noting that if the adversary injects a message right after the second step, the malicious message would disrupt j's communication to k, while k is cooperatively jamming. In this case, the adversary may not be caught by the identity checking, but more loss will be observed at k. □

6.2. A Proof for BrdSec(j ⟨ N_j): s

THEOREM 6.2. *If all nodes in N_j are legitimate, {BrdSec(j ⟨ N_j): s} satisfies the confidentiality property: j delivers s to all its neighbors N_j without revealing the content of s.*

PROOF. Following the proof for $UniSec(j \rightarrow k)$, we know that $\forall k \in N_j$, j delivers the initial jamming sequence securely to every node in N_j. Therefore, only nodes in N_j can decode the subsequent broadcast message. Also, a particular trusted node k is guaranteed to perform cooperative jamming according to the definition of $UniSec$, s is then protected. One thing to note is that if some message, for example, s_{m-1}, is blocked/lost in the middle, a legitimate receiver may not be able to decode the next message because it does not know how to compute its next jamming sequence, that is, $\gamma_{j,m}$. In that case, retransmission should be applied. □

6.3. A Proof for Authentication via Certificates

THEOREM 6.3. *The protocol for authentication via certificates satisfies the following properties: (i) if j and k have common nodes in their trusted base, they achieve mutual authenticatation; and (ii) if no legitimate node in the system knows both j and k, they cannot authenticate each other.*

PROOF. We prove this theorem in two cases: (i) if they have common nodes in a trusted base: according to Lemma 2.2, j's message "T_j" is delivered to k in an authenticated way, and k knows that it must be sent from j according to Lemma 2.3. By the same token, j knows "$T_k \cap T_j$" must be sent from k. According to the system model, a node can discover another neighbor's trusted base, therefore, nodes in the trusted base can verify whether j or k are lying. On the other hand, if j and k are legitimate, j and k can mutually discover their common trusted nodes, from which they can authenticate each other by the transitive property.

(ii) If they do not have any trusted node in common, the first step fails according to Lemma 2.3 because k does not trust j at the first place because no one who k trusts can introduce j. □

6.4. A Proof for One-Hop Anonymity for Unicast

THEOREM 6.4. *The anonymity protocol satisfies the following properties: (i) message s is delivered to k; and (ii) message s does not reveal the sender j's identity.*

PROOF. According to Lemma 2.2, j believes that his identity will be hidden by l's jamming. In the second step $j \langle\!\sim_{\gamma} l$: $f(s)$, the source of $f(s)$ cannot be learned by others because l performs a jamming during j's transmission. In the third step, l shouts out s to k without revealing the source of s. Meanwhile, j checks whether s is being transmitted by l by eavesdropping on the channel. After receiving s, k may recall that someone has shouted out a message with l's help, but it cannot figure out who the sender is. This achieves anonymity. We discuss this protocol in the presence of the following threats.
Eavesdrop: since there is no intention to secure the message, it is okay if all the messages in this protocol are being overheard. It is worth noting though that eavesdropping during the second step does not help to figure out j's identity because the overheard message is a combination of j's message and l's jamming message. The signature of this combination is certainly different from j's signature received at any eavesdropper.
Block: if the control message "ready to help" is blocked, then j is not willing to send anything anonymously to k because he does not find any helper. If other messages are blocked, retransmission may be needed. However, blocking any message in the protocol does not reveal j's identity.
Replay: replay the first message will fail according to Lemma 2.3. Replaying the next two messages is possible, however, it does not violate the anonymity requirement.
Injection: if the injection results in more loss, retransmissions may be applied. If the adversary injects any bogus message in any step, it does not hurt the system because any message may be allowed in a system with the anonymity requirement only. However, it does not make the protocol weaker because j's identity is still protected by the jamming signal from a trusted neighbor (i.e., l in the protocol). □

6.5. The Soundness and Completeness of the Zero Knowledge Alternative

LEMMA 6.5. *The zero knowledge alternative is sound in the sense that if a node owns its private credential, it can prove its identity in the zero knowledge protocol.*

PROOF.
If j owns s_j, since $G(n, s_j)^e \equiv u_j^e \cdot r_j^{e\cdot\theta(c_j,n)} \equiv j \cdot c_j^{\theta(c_j,n)}\ (mod\ w)$
and $g_j^e \equiv p^e \cdot r_j^{e\cdot\theta(c_j,z)} \equiv b \cdot c_j^{\theta(c_j,z)}\ (mod\ w)$,
j always succeeds in convincing k, therefore, this algorithm is sound. □

LEMMA 6.6. *The zero knowledge alternative is complete in the sense that if a node does not own the private credential, the probability that it can succeed in the zero knowledge protocol is low.*

PROOF. In order to prove the completeness, we need to show that anyone who doesn't know the secret can only succeed in convincing the verifier with small probability. Consider an adversary who wants to impersonate j, in order to launch a successful attack, the adversary needs to commit $(\tilde{c}_j, \tilde{g}_j)$, guess the challenge $\tilde{n}$, and a response $\tilde{G}$ such that: (1) $\tilde{G}^e \equiv j \cdot \tilde{c}_j^{\theta(\tilde{c}_j,\tilde{n})}\ (mod\ w)$, and (2) $\tilde{g}_j^e \equiv b \cdot \tilde{c}_j^{\theta(\tilde{c}_j,z)}\ (mod\ w)$.
Let us focus on Eq. (2) first. Given $\tilde{c}_j$, the probability of guessing $\tilde{g}_j$ correctly is low because computing modular roots is well considered as a hard problem. On the other hand, given $\tilde{g}_j$, the probability of guessing $\tilde{c}_j$ is low because θ is a one-way function. Even if $\tilde{c}_j$ and $\tilde{g}_j$ meet the condition (2), since the challenge $\tilde{n}$ varies at different time, the probability of guessing the response $\tilde{G}$ correctly to satisfy condition (1) is low because computing modular roots in condition (1) is hard. □

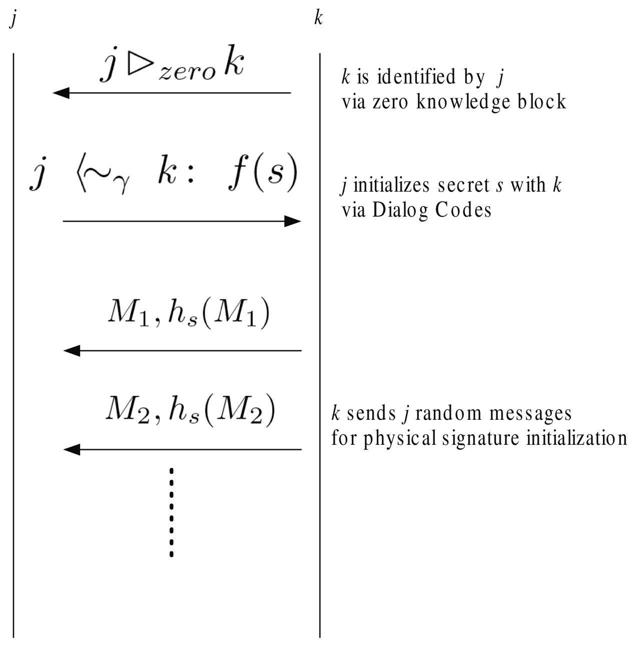

j
k
$j \rhd_{zero} k$
k is identified by j via zero knowledge block
$j \langle\sim_\gamma k: f(s)$
j initializes secret s with k via Dialog Codes
$M_1, h_s(M_1)$
$M_2, h_s(M_2)$
k sends j random messages for physical signature initialization

Fig. 4. An example of initializing the entire network.

REFERENCES

Arora, A. and Sang, L. 2009. Dialog codes for secure wireless communications. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'09)*. 13–24.

Danev, B. and Capkun, S. 2009. Transient-Based identification of wireless sensor nodes. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'09)*.

Fiat, A. and Shamir, A. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings of the Annual Cryptology Conference (CRYPTO'86)*.

Fischer, R. J. and Green, G. 2003. *Introduction to Security*. Butterworth-Heinemann.

Kansei Testbed. 2012. http://cast.cse.ohio-state.edu/kansei/

Katti, S., Gollakota, S., and Katabi, D. 2007. Embracing wireless interference: Analog network coding. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*.

Lai, L., Gamal, H. E., and Poor, H. V. 2008. The wiretap channel with feedback: Encryption over the channel. *IEEE Trans. Inf. Theory* 54, 11, 5059–5067.

Parno, B. 2008. Bootstrapping trust in a 'trusted' platform. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security (HotSec'08)*.

Patwari, N. and Kasera, S. 2007. Robust location distinction using temporal link signatures. In *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom'07)*.

Perrig, A. and Tygar, J. D. 2003. *Secure Broadcast Communication in Wired and Wireless Networks*. Kluwer Academic.

Rivest, R., Shamir, A., and Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM 21*, 2, 120–126.

Sang, L. and Arora, A. 2008a. Capabilities of low-power wireless jammers. Tech. rep. OSU-CISRC-5/08-TR24, Ohio State University.

Sang, L. and Arora, A. 2008b. Spatial signatures for lightweight security in wireless sensor networks. In *Proceedings of the IEEE InfoCom Miniconference*.

Sayeed, A. and Perrig, A. 2008. Secure wireless communications: Secret keys through multipath. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'08)*.

Shamir, A. 1984. Identity-Based cryptosystems and signature schemes. In *Proceedings of the Annual Cryptology Conference (CRYPTO'84)*.

Tekin, E. and Yener, A. 2008. The general Gaussian multiple access and two-way wire-tap channels: Achievable rates and cooperative jamming. *IEEE Trans. Inf. Theory 54*, 6, 2735–2751.

Wyner, A. D. 1975. The wire-tap channel. *Bell Syst. Tech. J. 54*, 8, 1355–1387.

Received February 2010; revised August 2010; accepted November 2010