Establishing Authenticated Channels and Secure Identifiers in Ad-hoc Networks*

Bartlomiej Sieka[1,2] and Ajay D. Kshemkalyani[2]
(Corresponding author: Bartlomiej Sieka)

Semihalf, Cracow, Poland (Email: tur@semihalf.com)[1]
Computer Science Department, University of Illinois at Chicago[2]
Chicago, IL 60607-7053, USA (Email: ajayk@cs.uic.edu)

(Received Oct. 27, 2005; revised and accepted Dec. 31, 2005 & Mar. 12, 2006)

Abstract

In this work, we describe a method for establishing authenticated channels in a wireless ad-hoc network. The presented protocol is fully self-organized, and it also provides an identification framework. The two main contributions are (1) a fully self-organized protocol that establishes an authenticated communication channel between nodes of a wireless ad-hoc network and (2) a secure identifier framework that is resilient to impersonation. The authenticated channel provided by the protocol can then be used to establish a secret communication channel between nodes. The authentic identifiers established can be used to associate network (and upper) layer identifiers to prevent spoofing. They can also serve as a reliable basis for reputation management protocols. The paper analyzes the proposed approach in terms of security and complexity (both communication and storage).

Keywords: Ad-hoc networks, authentication, key establishment, secure identifiers

1 Introduction

1.1 Preliminaries

The problem of creating a secure communication channel in a wireless ad-hoc network is important. The existence of such a channel between two nodes is the foundation of all secure interaction between them. A secure channel is defined as having the following attributes.

- The channel is immune to eavesdropping (secrecy).
- Any alteration to a transmitted message is detected by the receiver, and the communication parties are mutually authenticated (integrity).
- The channel operation can not be disrupted by a malicious entity (availability).

*A preliminary version of this work appeared in [22].

We consider only the first two attributes – channel secrecy and channel authentication. Note that the problem of providing an authenticated channel underlies the problem of providing a secret channel. In most practical scenarios, channel secrecy is provided by the symmetric cryptography. This approach requires that the two communicating parties share a common secret (the secret key) and thus a method of providing that secret key material is necessary. This is achieved by key establishment protocols which can be classified as follows.

Key transport. One party generates the key and transports it securely to the other party. This very need to securely transport the key makes this approach not suitable for ad-hoc networks.

Key agreement. Both communicating parties must contribute to the generation of the secret key. It is then used to secure transmission between them.

Current key agreement protocols typically rely on the existence of a small-size authenticated channel ([4]). For example, consider a scenario where public key cryptography is used. Here, the authenticity of the public key needs to be verified by some out-of-band communication channel. Even though the size of such a out-of-band channel is very small compared to the size of the authenticated (and possibly secure) channel that results from the use of cryptographic methods, observe that such a channel must itself be authenticated. The main purpose of the out-of-band communication is to authenticate the identifier of the communication party, i.e., to know the identity of the sender. Thus it is important to provide an authenticated channel before providing other services and protocols, such as secure routing, in ad-hoc networks.

1.2 State of the Art

The existing protocols to establish secure channels can be classified into three approaches.

1) *Protocols using prior context.* It is easier to secure ad-hoc networks if the nodes share some *prior context* before the network operation begins. A prominent family of protocols using this approach are the protocols that assume an off-line secret key predistribution phase. Such protocols have received a lot of research attention recently, especially in the context of sensor networks ([10, 12, 13, 14, 15, 19]). As this approach requires that the nodes have a shared prior context before the deployment, it is not always practical and we seek to provide a full self-organized solution that is free of this drawback.

2) *Trusted third party protocols.* These protocols rely on the existence of a *trusted third party (TTP)*, such as a Certificate Authority, a base station, or a designated node. This approach implies centralization of vital network services and thus is not well suited for the ad-hoc (peer-to-peer) setting.

3) *Self-organizing protocols using out-of-band channels.* The most natural approach to take for ad-hoc networks is *self-organization.* In this approach, there are no special nodes, no infrastructure, no centralized configuration point, no shared prior context. Unfortunately, many protocols proposed have a limiting assumption, that there exists an *out-of-band authenticated communication channel.* A widely referred work on using out-of-band channels to initialize secrets is due to Stajano and Anderson [23]. Balfantz et al. [3] propose a pre-authentication protocol that can be used to bootstrap secure communication in an ad-hoc wireless network. They use a *privileged side channel* to perform the out-of-band preauthentication protocol. They use *location-limited channels*, meaning that they are available in close physical proximity of communicating parties.

   A very interesting thread of research is presented by the group from the Swiss Federal Institute of Technology at Lausanne ([7, 9, 17]). They propose the use of public certificate chains for authentication with probabilistic guarantees. However, it seems difficult to *bootstrap the initial certificate repository of a node without an authenticated channel* of some sort. Also, Capkun and Hubaux in [8] assume that there exist *security associations* for a percentage of nodes, and then propose BISS – an optimization to an existing routing protocol that results in a secure routing scheme. Authors in [11] propose a security scheme based on identity-based and threshold cryptography. They *assume that authentic identifiers have been bootstrapped* in the network prior to the operation of their scheme. Khalili et al. in [18] propose the use of identity-based cryptosystems with threshold cryptography for key distribution in ad-hoc networks. Their ID-based approach assumes that *there exists a public master key known to a set of nodes.* It appears to be difficult to perform this step, especially so that the nodes can be assured about the authenticity of that public key. An interesting proposal advocating the use of the *tamper-proof hardware* is the UGSP protocol described in [2]. The authors propose a key establishment protocol that requires the user to interface a hardware token with the mobile device that he wishes to use for communication.

1.3 Contribution

In an ad-hoc peer-to-peer setting, an authenticated channel is a necessary prerequisite for key establishment protocols, that are used to build secure communication channels. The authenticated channel can also be used as a building block for a secure routing algorithm. Once a secure routing is in place, we can implement any other network services.

The *main contribution* of this paper is a self-organized protocol to establish authenticated channels in a peer-to-peer network, without assuming any network services and using only an unsecured and untrusted channel. This protocol has none of the drawbacks of the three categories of protocols (1)–(3) described in Section 1.2, and meets the following *requirements* R1 – R3.

R1. There is no requirement to share a prior context.

R2. No trusted third party is needed.

R3. No out-of-band authenticated channels, privileged side channels, prior security associations, prior bootstrapping of authenticated identifiers, a priori known public master key, or tamper-proof hardware are used.

Although the protocol is simple, its importance stems from the fact that it is the first general protocol to satisfy requirements R1 – R3, and it can serve as the base for the development of further protocols exhibiting *all* the requirements R1 – R3. The protocol to implement the authenticated communication channel uses an unsecure communication medium, as shown in Figure 1. Any network service, including secure routing and forwarding, can use such authenticated channels. However, even with authenticated channels, the authenticity of communicating parties must be considered in the context of their identities. Therefore, as part of our solution, we also address the issue of the identity model of the network, proposing the use of *secure identifiers.* This is the *second contribution* of this paper. We propose to use the hash of the public key for the identifier of a node. Although this idea has been used before (e.g., [20, 21]), the hash of the public key as a secure identifier has always been used in the context of a specific protocol. For example, [21] employs a very similar method to secure binding update messages in the mobile IPv6. However, that proposal is very closely tied to a very particular application. On the other hand, our approach does not make strict assumptions about the properties of

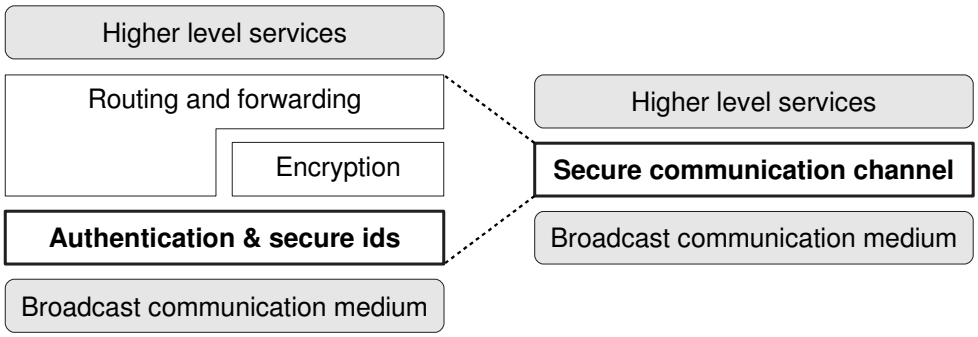

Figure 1: Layering diagram showing the placement of the authentication and secure identifiers protocol

adjacent protocol layers, thus being suitable for incorporation in many contexts. We solve the problem of providing secure identifiers in a *protocol-independent manner*. *Any* higher layer protocol (see Figure 1), including routing and forwarding, can use these secure identifiers.

Bobba et al. [5] presented the cyclic dependency between secure routing and security services, as shown in Figure 2. They then proposed a self-organized approach for bootstrapping security for the routing layer. Their approach breaks the cyclic dependency by proposing a secure routing protocol that does not depend on any security services. Thus, their approach removes edge (1) to remove the cyclic dependency. Their solution is embedded in the routing protocol and does not establish secure identifiers for the higher layers. We take a complementary route to breaking the cyclic dependency: we aim at removing dependency edge (2) in Figure 2, by showing how to implement security services without relying on the routing. Bobba et al. [5] had stated that "Removing dependency (2) would be impractical because it would require that the nodes implementing security services be reachable by all other nodes in the network by a fixed set of routes." However, our approach (see also [22]) does remove dependency (2) because we do not rely on the existence of a routing protocol in our approach; hence the nodes need not be reachable by a set of routes.

Section 2 gives a formal statement of the problem. Section 3 presents the solution protocol for a peer-to-peer ad-hoc wireless network. Section 4 performs a security analysis. Section 5 gives the complexity analysis. Section 6 discusses further aspects of the solution. Section 7 gives concluding remarks.

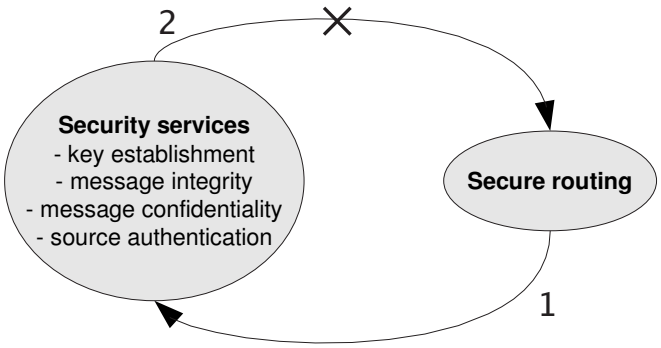

Figure 2: Dependency cycle between secure routing and security services. Figure is adapted from [5] to reflect our approach of removing dependency (2) to break the cycle.

2 Objectives

Our goal is to devise a means for authenticated communication within a peer-to-peer ad-hoc network. As the authenticity of communication is very closely related to the identity of communicating parties, we also aim at a method for providing network identifiers.

The problem is formalized as follows. Given a set of nodes, we want them to be able to establish an authenticated network (AN for short). More formally, for any given node j in the AN we want the following property.

$$Key\ Property:\ \forall i \in AN, PK_i^j = PK_i^i. \qquad (1)$$

PK_i^j is the public key of node i as known by node j. Note that PK_i^i can be seen as the authentic public key of node i. In other words, the above property states that all nodes in the AN know the authentic public keys of all other nodes in the AN. The authenticity of the public key must be considered in the context of the entity that is in the possession of the corresponding private key. Hence we define PK to be an authentic public key of node i if and only if (a) node i possesses the corresponding private key and (b) the identity of node i can be verified to be bound to that key. We do not require that the node i generates both the public and private keys. In case they are generated by some other party, we implicitly assume that there exists a trust relationship between the key generator and that node. It should be noted that this assumption is very subtle and should be carefully verified when considering a larger context.

The problem stated above is essentially that of bootstrapping the network. We assume that network layer functions are not yet available, i.e., nodes do not forward

packets and do not implement any routing algorithm.

3 The Key Agreement Protocol

To determine a suitable identity model we emphasize a quote from [18]: "in an ad-hoc network there may be no *a priori* reason to distinguish between the different nodes" (Section 2.2, last sentence). In this spirit we propose that the identifiers are self-appointed. We start by assuming that every node has a pair ⟨ private key, public key ⟩. We propose that the node identifier be a hash of the node's public key. For example, for node i we have $id_i = hash(PK_i)$ where id_i is the assumed identifier of the node, and PK_i is its public key. This approach has been described in [20, 21]. Observe that if the Key Property in Equation (1) holds, then all the nodes in the AN know each other's identifiers.

The following notation is used to describe protocol messages. When the contents of the message are relevant, we use $MSG(contents)$ to denote the message, where *contents* lists the actual fields of the message. When we want to refer to a given message more generally, we use the shorter `TYPE` notation, where `TYPE` is a place-holder for the type of message in question (e.g., `JOIN` or `ACCEPT` or `UPDATE`). The type of message is usually the first field of its contents – it can be an integer number or a string and must be unique across different message types. PK_i and SK_i are used for node i's public and private key, respectively. To indicate that the integrity of the message M is protected by a digital signature, we use the notation $S_{SK}(M)$ where SK is the private key used for signing. The symbol $\rightleftharpoons$ denotes radio broadcast when describing the flow of messages.

The protocol uses the following data structures. Each node maintains a key table that gives the mapping between the identifiers and public keys, as well as some additional information. Each node j in the network maintains the following data for each other node i.

- id_i: Node identifier. In our approach, the identifier is a hash value of the public key of the node, i.e., $id = hash(PK)$. Note that the node identifier is a fixed-length sequence of bits.
- PK_i: Public key of the node.
- seq_i: An integer that gives the sequence number copied from the last `JOIN` message from i.
- $time_i$: An integer that gives the local time when the most recent message from node id_i was seen.

Node j initializes its key table with one row containing its identifier and public key. Note that both the seq_j and $time_j$ are not relevant in this case. Table 1 illustrates the format of a node's key table. The number of entries in the key table of node j is denoted as N_j.

The protocol description uses variables $KeyTable_j$ and $KeyTableDelta_j$. $KeyTable_j$ denotes two columns of node j's key table: the PK column and the seq column. $KeyTableDelta_j$ denotes the set of those (PK, seq) entries from j's key table, that have been modified since the last time a `UPDATE` message was sent by node j.

Table 1: A key table maintained by a node j

node	id	PK	seq	time
1	id_1	PK_1	seq_1	$time_1$
2	id_2	PK_2	seq_2	$time_2$
...	...	...	...	...
j	id_j	PK_j	seq_j	$time_j$
...	...	...	...	...
N_j	id_{N_j}	PK_{N_j}	seq_{N_j}	$time_{N_j}$

The proposed protocol to build an AN is outlined in Figure 3 and is organized around the following two main scenarios.

Node Join: A node outside of an AN wants to join AN. This node may or may not be a member of some other AN. If it is a member of another AN, then the joining scenario is equivalent to a merge of the two networks. The entire network AN can be viewed as having been formed from a succession of subnetwork merges. In each merge, one node initiates the protocol for the joining of two (or more) sub-networks by sending a `JOIN` message. The nodes that receive a `JOIN` respond with an `ACCEPT` message. This is illustrated in Figure 4. A single `JOIN` may trigger more than one `ACCEPT`, one from each node within radio coverage. Note that the nodes that send the `ACCEPT` messages in response may belong to different ANs in the general case. Such cases are analyzed in Section 5.

Let the sub-network of the node that initiates the `JOIN` be denoted as AN_{join}, and let the sub-network(s) that accept this initiated merge be denoted as AN_{accept}. When $|AN_{join}| = 1$, we have the special case when a single node wants to join an existing AN on awakening or recovering.

Key Update: The contents of the key table change over time. Entries can be added as a result of receiving the `JOIN`, the `ACCEPT`, or the `UPDATE` message. Upon a change in the key table, the node should notify others by broadcasting the `UPDATE` message itself. For node j the following message should be used.
$j \rightleftharpoons MSG(\texttt{UPDATE}, id_j, KeyTableDelta_j, S_{SK_j}(\texttt{UPDATE}, id_j, KeyTableDelta_j))$.

The protocol models the following seven events: *Send* `JOIN`, *Receive* `JOIN`, *Send* `ACCEPT`, *Receive* `ACCEPT`, *Send* `UPDATE`, *Receive* `UPDATE`, and *Key Timeout*. The first six correspond to the send and receive of the three types of messages used, and the seventh corresponds to a timeout for maintaining the soft-state key table. *Send* `JOIN`,

When node i wants to join another AN:
it broadcasts the JOIN message:
$i \rightleftharpoons MSG(\texttt{JOIN}, id_i, seq_i, KeyTable_i, S_{SK_i}(\texttt{JOIN}, id_i, seq_i, KeyTable_i))$.

When node $j \in AN$ receives the JOIN from node i and determines that i can join the AN:
j enters i's data into its key table and broadcasts the ACCEPT message:
$j \rightleftharpoons MSG(\texttt{ACCEPT}, id_j, seq_i, PK_i, KeyTable_j, S_{SK_j}(\texttt{ACCEPT}, id_j, seq_i, PK_i, KeyTable_j))$.
Then j broadcasts the UPDATE message:
$j \rightleftharpoons MSG(\texttt{UPDATE}, id_j, KeyTableDelta_j, S_{SK_j}(\texttt{UPDATE}, id_j, KeyTableDelta_j))$.

When node i that has sent a JOIN receives the corresponding ACCEPT message:
i updates its key table.
Then the node broadcasts an UPDATE message:
$i \rightleftharpoons MSG(\texttt{UPDATE}, id_i, KeyTableDelta_i, S_{SK_i}(\texttt{UPDATE}, id_i, KeyTableDelta_i))$.

When node j inside the AN receives a UPDATE message:
j updates its key table using $KeyTableDelta$.
If new entries are added to its key table,
the node broadcasts the UPDATE message:
$j \rightleftharpoons MSG(\texttt{UPDATE}, id_j, KeyTableDelta_j, S_{SK_j}(\texttt{UPDATE}, id_j, KeyTableDelta_j))$.

Figure 3: Outline of protocol to establish authenticated channels

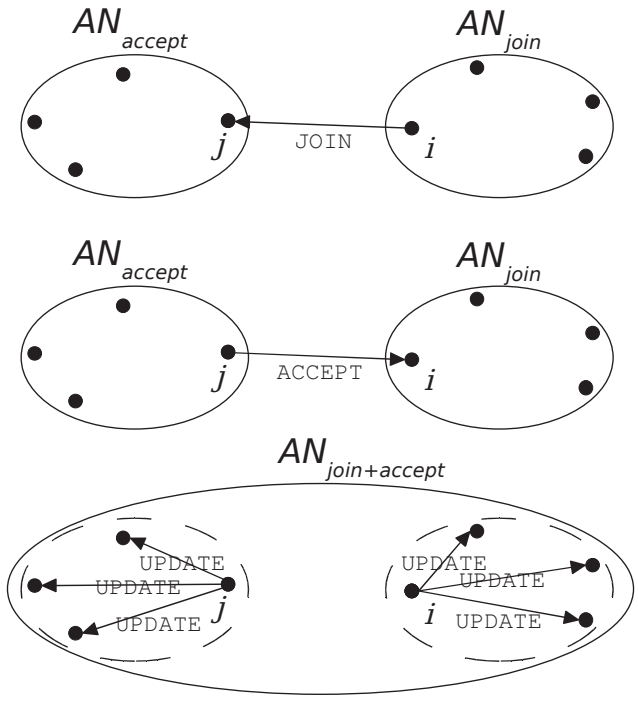

Figure 4: A graphical depiction of the main steps of the JOIN scenario. Note that all the messages are broadcast – arrows are only used to indicate the actual recipients.

Receive JOIN, *Receive* ACCEPT, *Receive* UPDATE, and *Key Timeout* trigger other actions. The processing for each of these seven events is described below. Using a reactive interrupt-driven formulation, this processing is summarized in Figure 3.

1) *Send* JOIN: This event pertains to a node in AN_{join}. When a node i wants to join another AN, it should generate a pair of keys (PK_i, SK_i), where PK_i is the public key and SK_i is the corresponding secret key and then broadcast the JOIN message. If it has already generated this pair before, but is now joining a new AN due to mobility or other reasons, it does not have to regenerate a different key pair. The seq_i field of the message is a sequence number that is guaranteed to be increasing with time for a given node and the id_i is the hash of the public key generated.

2) *Receive* JOIN: This event pertains to a node that is not a member of AN_{join}. When a node j receives a JOIN message, it should first verify the validity of the digital signature of the message. If the signature is invalid, the message should be discarded and no further action should be taken. If the signature is valid, j should compute the hash value of the public key $id_i = hash(PK_i)$ and check if there exists an entry with id_i for node i in its (i.e. j's) key table.

   a. If the id_i entry does not exist, then the new entry should be added with the computed id_i and both PK_i and seq_i values copied from the JOIN message. The $time_i$ field should be set to node j's local time. The node should then broadcast the ACCEPT message, and also broadcast the UPDATE message (see event 5).

b. If the id_i entry does exist, let k be the index of that entry in the key table; hence $id_i = id_k$. Also the corresponding public key and sequence number are denoted by PK_k and seq_k, respectively. There are three cases.

i. $PK_i \neq PK_k$: There is a collision in the hash function. The `JOIN` message should be discarded. Dealing with collisions is discussed in Section 6.

ii. $PK_i = PK_k$ and $seq_i < seq_k$: This might indicate an attempt to mount a replay attack. (Note that the `JOIN` messages are not subject to the regular network routing and forwarding, hence this case does not indicate routing loops or other network-layer problems.) This can also mean that node i sequence counter has wrapped around. The `JOIN` message should be discarded.

iii. $PK_i = PK_k$ and $seq_i \geq seq_k$: This may indicate that the node i is sending spurious `JOIN` messages. The sequence number seq_k should be updated to seq_i, the `ACCEPT` message should be broadcast, and then the `UPDATE` message should be broadcast (see event 5).

3) *Send* `ACCEPT`: This event pertains to a node that is not a member of AN_{join}. On receiving a `JOIN` message from node i (Event (2)), when a member node j determines that i can be admitted to its AN without an identifier conflict, it broadcasts the `ACCEPT` message. The $KeyTable_j$ field of the message should contain all the (PK, seq) pairs from the key table of node j.

4) *Receive* `ACCEPT`: This event pertains to the specific node in AN_{join} that sent the `JOIN` message. After broadcasting the `JOIN` message, when the node in AN_{join} receives a corresponding `ACCEPT` message, it is considered to be a member of the $AN_{join+accept}$ network. The node should check the signature of the message and drop the message if the check fails. If the signature is valid, the node should add entries from the $KeyTable$ field of the `ACCEPT` message to its key table. Then the node broadcasts an `UPDATE` message so that any other AN also within its range can learn of AN_{accept}.

A node that did not send the `JOIN` message should drop any `ACCEPT` message received.

5) *Send* `UPDATE`: This event pertains to a node that is a member of the AN_{join} or AN_{accept}. New entries are added to the node's key table in the following cases.

Event 2a: node in AN_{accept} sends `UPDATE`.

Event 2(b)iii: node in AN_{accept} sends `UPDATE`.

Event 4: node in AN_{join} sends `UPDATE`.

Event 6: node in AN_{join} or AN_{accept} sends `UPDATE`.

When new entries are added to the node's key table, the node should broadcast the `UPDATE` message. The $KeyTableDelta$ field of the message should contain all the (PK, seq) pairs that have been updated since the last time the `UPDATE` message was sent.

6) *Receive* `UPDATE`: This event pertains to a node that is a member of AN_{join} or AN_{accept}. When a node receives the `UPDATE` message, it should check the signature of the message. The message should be dropped and no further action should be taken if the signature is invalid, otherwise the node should add entries from the $KeyTableDelta$ field to its key table. It then executes actions associated with *Send* `UPDATE`, see event (5). A node outside of the AN should drop the `UPDATE` message.

7) *Key Timeout:* Each node should maintain a timestamp associated with every entry in its key table (field *time* in the key table). Node i should update the timestamp to its current time for entry j every time it receives a message sent by node j. Note that the message need not be addressed to node i. An entry should be deleted from the key table if the timestamp is older than a specified tunable value. The default can be set to a high value to minimize overhead under normal operation. This timeout mechanism implements a soft-state key table.

4 Security Analysis

The protocol assumes no prior context shared between nodes and succeeds in allowing the nodes to perform some mutually desired interaction. This is similar to meeting a stranger who introduces himself as "X". One cannot trust the stranger but needs to build trust over time, associating its level and other attributes with the name "X". If over time, all the interactions allow us to achieve our goals, and the person has been well-behaving until then, we do not care much about what that person's real name is. The initial communication phase and the subsequent process of associating trust are important.

Analogously, in the absence of prior context and lack of infrastructure, we cannot do better that to trust the other communication party with what he says. In our protocol, the initial trust is restricted to the identity (and the public key), as communicated by the other party. Under this assumption, a *man-in-the-middle* attack looses its meaning. Consider a malicious node that is in between us and some other node and just relays that node's communications to and from us. Since all the messages are signed, the attacker has no other choice than to use his own private key and the corresponding public key as the identifier (otherwise messages we receive would fail the integrity check). From our perspective, all the messages appear as coming from the attacker, since his identity can be bound to the message by checking of the message signature. If such interactions are satisfactory to us, then it

is really irrelevant who the sender of the message is.

In our identity model, impersonating a node is equivalent to being able to generate a signature using that node's private key. The facts that each message carries a digital signature, and that a node's identity is bound to its public key, make *impersonation* attacks impossible. Further, the *replay* and *reflection* attacks against the joining phase are thwarted by the use of sequence numbers.

All practical applications will require the identifier to be bound to some other higher-level information (e.g., a person name, an IP address, a MAC address, an organization name). The existence of an authenticated channel provided by our protocol allows those higher-level associations to be established securely using cryptographic techniques (both symmetric and public key schemes can be used).

A specific example of a higher level data that can be associated with the identity of the node is its reputation. Mechanisms for managing reputations, such as those for peer-to-peer and ad-hoc networks ([1, 6, 16, 24]), can be combined with our protocol to provide mechanisms to increase (or decrease) the trust level between nodes. To our knowledge, the reputation management schemes rely on the existence of a secure outside communication channel, which our protocol provides.

In the *Sybil* attack ([25]), a malicious entity gains control of multiple network identities and leverages that control to perform some unauthorized action. Sybil attack is mostly applicable against reputation management protocols, quorum-based systems and systems where certain functions require cooperation of a given number of nodes (e.g., threshold cryptography). Consider now the Sybil attack in the context of the higher-level information mentioned above. In principle, there is nothing that would prevent an entity from generating multiple public, private key-pairs and thus assuming multiple identities. This is due to the assumed characteristics of the communication medium, i.e. its unreliability and its broadcast nature. Thus, mechanisms on higher levels must be used to addressing the Sybil attack threat, for example, reputation management approaches mentioned above.

All the attacks considered so far (*man-in-the-middle*, *impersonation*, *replay*, *reflection*, *Sybil*) are *active* attacks. They require the potential attacker to perform some action, to take part in the communications by sending messages. Let us now consider a *passive* attack. In such a scenario, the attacker is only listening to the communications without performing any actions. In particular, consider the passive eavesdropping attack, i.e., discovering the contents of communications that are intended to be secret, except for the authorized parties. Observe that the proposed protocol results in nodes knowing each others' authentic public keys. With this, the nodes can safely set up a communication channel protected by symmetric cryptography. Thus the proposed protocol provides resilience to passive eavesdropping attacks.

The approach presented in this work exhibits good security properties, as it is immune to various attacks. It should be noted that most of the work presented in the literature (see Section 1.2) presents comparable guarantees. With only few exceptions (e.g., [18] vulnerable to the man-in-the-middle attack), the prior protocols are also safe against the attacks analyzed above. However, the approach presented here provides security with less restrictive assumptions, meeting requirements R1 – R3.

5 Complexity Analysis

Let N be the number of entries in the sender's key table and M be the number of fresh entries in the sender's key table. The sizes of the data fields and messages used in the protocol are given in Tables 2 and 3. We use notation |`JOIN`| to denote the size of the `JOIN` message; similarly for the other messages. N is a parameter for `JOIN` and `ACCEPT` messages, M is a parameter for `UPDATE` messages. Assuming a 1536-bit public key, the number given in Table 2 is for the compressed public key, used in transmission. Note that as we use hashes of the public keys as the identifiers, we save on message size for the higher layer protocols. To analyze the protocol complexity, we assume that the nodes are static, and that there is steady traffic among the nodes after bootstrapping. The latter assumption rules out churn in the key tables by preventing aging of entries.

Table 2: Sizes of data fields

data field	size (bytes)
id	4
PK	64
seq	4
S_{SK}	97

We define a *joiner* as a node from AN_{join} that initiates the protocol and an *acceptor* as a node from AN_{accept} that receives the `JOIN` message and responds with `ACCEPT`. Usually i will be used to describe a joiner and j will be used to describe the acceptor. Let B_i denote the set of nodes in the neighborhood of node i (i.e., the nodes within i's radio range) at the time i individually joins another AN. Let $|B_i|$ be denoted by b_i. We assume that there is one joiner in each merge operation. This does not lose generality because we can simply require a singleton node that is not part of a larger AN but wants to join, to remain silent until it hears no bootstrapping messages. Figure 5 depicts merge scenarios described in the following sections.

5.1 Simple Scenario

Consider a *node join*, where a single node i (i.e., $AN_{join} = \{i\}$) wants to join the AN_{accept}. Here we assume a single AN within radio range of the joiner. Node i triggers

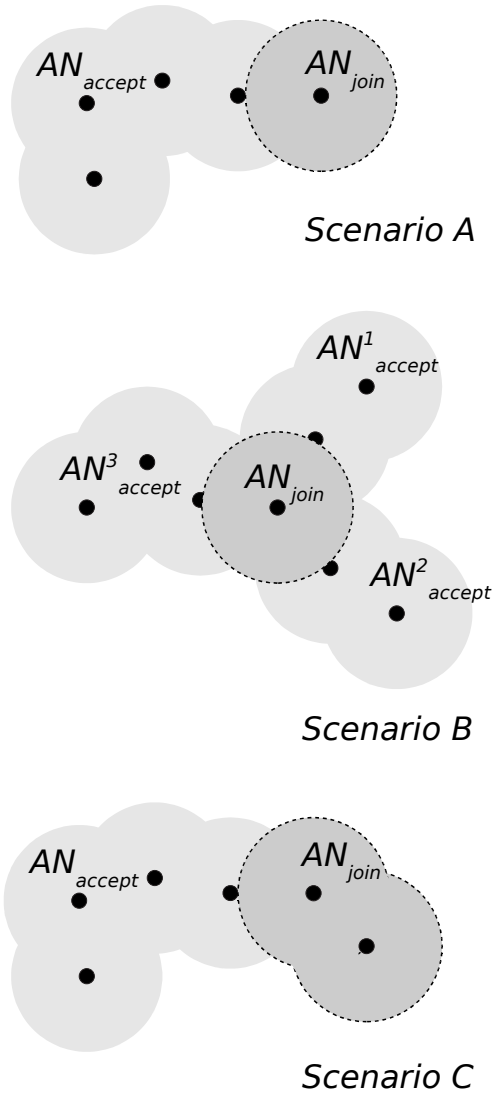

Figure 5: Three merge scenarios. Scenario A is a simple case where a single node joins an existing AN (Section 5.1). Scenario B shows a single node joining multiple ANs (Section 5.2), and Scenario C depicts two ANs merging together (Section 5.3).

Table 3: Sizes of the messages used

message			size (bytes)
`JOIN`	$1+4+4+(64+4)N+97$	$=$	$68N+106$
`ACCEPT`	$1+4+64+(64+4)N+97$	$=$	$68N+166$
`UPDATE`	$1+4+(64+4)M+97$	$=$	$68M+102$

the protocol by broadcasting the `JOIN` message; the result is $AN_{join+accept}$ that is one node larger in size than AN_{accept}. The `JOIN` message triggers b_i `ACCEPT` messages. Each of the b_i nodes also broadcasts an `UPDATE` message, and eventually all nodes in AN_{accept} receive an UPDATE and broadcast an `UPDATE`. Thus, there are $|AN_{accept}|$ `UPDATE` messages. Finally, node i also broadcasts an `UPDATE` after receiving the first `ACCEPT`. The number of messages broadcast for this merge is

$$1 + b_i + (|AN_{accept}| + 1). \tag{2}$$

The number of bytes transmitted for this merge is

$$1 \cdot |\texttt{JOIN}(1)| + b_i \cdot |\texttt{ACCEPT}(|AN_{accept}|)| + |AN_{accept}| \cdot |\texttt{UPDATE}(1)| + 1 \cdot |\texttt{UPDATE}(|AN_{accept}|)|.$$

5.2 Merging with Multiple ANs

A single joiner node has x disjoint ANs in its radio range. The ANs are disjoint because they are outside each other's radio range. Assuming the unit disk radius model, observe that $x \leq 6$ as at most six equilateral triangles can have their apex at the center of the circular radio coverage of the joiner. Let the adjacent ANs be identified by superscripts. There will be one `JOIN` and b_i `ACCEPT`s. The joiner will perform x `UPDATE`s. The number of `UPDATE`s performed in AN^y_{accept}, where $y \in [1, x]$, is given as $(|AN^y_{accept}|)(1 + (x - 1))$. For each node in each AN^y_{accept}, there is one `UPDATE` broadcast in response to the joiner's `JOIN` message, and each of the other $x - 1$ `UPDATE` broadcasts is triggered by the `UPDATE` broadcast by the joiner in response to the `UPDATE` the joiner received from other ANs. The number of messages broadcast for this merge is

$$1 + b_i + x + (\sum_{y \in [1,x]} |AN^y_{accept}| \cdot x). \tag{3}$$

The number of bytes transmitted for this merge, assuming each node in radio range that responds with an `ACCEPT` belongs to an AN of uniform size $|AN_{accept}|$, is given by

$$1 \cdot |\texttt{JOIN}(1)| + b_i \cdot |\texttt{ACCEPT}(|AN_{accept}|)| + \sum_{y \in [1,x]} |\texttt{UPDATE}(|AN^y_{accept}|)| + \sum_{y \in [1,x]} (|AN^y_{accept}|)(|\texttt{UPDATE}(1)| + \sum_{z \in [1,x] \setminus y} |\texttt{UPDATE}(|AN^z_{accept}|)|).$$

The third term represents the byte count in the UPDATE messages broadcast by the joiner across its x broadcasts. In the fourth term, $\sum_{y\in[1,x]}(|AN^y_{accept}|)$ represents the count of nodes in all the x ANs which accept the JOIN of the joiner. Each such node broadcasts:

- one UPDATE because of the joiner joining the AN. The byte count of this UPDATE is UPDATE(1).
- one UPDATE for each of the other $x-1$ ANs, because the joiner receives and forwards the UPDATE$(|AN^z_{accept}|)$ from each other AN z that it joins. The total byte count of these UPDATE messages broadcast by this one node in AN^y_{accept} is $\sum_{z\in[1,x]\setminus y}|\texttt{UPDATE}(|AN^z_{accept}|)|$.

5.3 Merging Two ANs

Both AN_{join} and AN_{accept} have size larger than 1. This scenario can occur if AN_{join} is mobile and the joiner moves within range of AN_{accept}. We assume there is a single joiner node in AN_{join}. The number of messages broadcast for this merge is

$$1 + b_i + (|AN_{accept}| + |AN_{join}|). \quad (4)$$

The number of bytes transmitted for this merge is

$$1 \cdot |\texttt{JOIN}(|AN_{join}|)| + b_i \cdot |\texttt{ACCEPT}(|AN_{accept}|)| + |AN_{accept}| \cdot |\texttt{UPDATE}(|AN_{join}|)| + |AN_{join}| \cdot |\texttt{UPDATE}(|AN_{accept}|)|.$$

5.4 Overall Bootstrapping Cost – Broadcasts

For each of the scenarios in Sections 5.1, 5.2, and 5.3, let n be $|AN|$ after the JOIN operation completes. For the scenario in Section 5.1, the total number of broadcasts $T(n)$ is expressible by the following recurrence relation derived using Equation (2).

$$T(n) = n + 1 + b_i + T(n-1). \quad (5)$$

For the scenario in Section 5.2, using Equation (3), the total number of broadcasts can be given by

$$T(n) \le 7 + b_i + \sum_{y\in[1,6]} |AN^y_{accept}| \cdot 6 + \sum_{y\in[1,x]} T(n_y), \quad \text{where} \sum_{y\in[1,x]} n_y = n-1. \quad (6)$$

For the scenario in Section 5.3, using Equation (4), the total number of broadcasts can be given by

$$T(n) = n+1+b_i+T(n_1)+T(n_2), \textit{where } n_1+n_2 = n-1. \quad (7)$$

In all three cases of Equations (5), (6), (7), the recurrence relations can be solved to show that

$$T(n) = O(n\cdot(n+1)/2 + \sum_{i\in[1,n]} b_i) = O(n^2).$$

5.5 Overall Bootstrapping Cost – Message Space

The logic to compute the total message space to bootstrap an AN uses the "number of key table entries broadcast" as a metric, and is as follows. The key table entry for node i enters each node's key table.

- When node j first learns i's key, it broadcasts an UPDATE and no more UPDATEs will contain i's key table entry.
- Node j broadcasts i's key table entry in an ACCEPT each time it receives a JOIN, and only *after* i's key table entry begins to exist in its own key table. The total number of JOINs received by j is bounded by $|AN|/2\cdot\bar{b}$, where $\bar{b}$ is the average of the values b_k.

Skipping the intermediate steps, we can show that the total number of times a node i's table entry is broadcast, which gives the message space cost for bootstrapping that node i's entry in the entire network, is bounded by

$$|AN| + \bar{b}\cdot|AN|/2 = |AN|\cdot(1+\bar{b}/2).$$

The total message space complexity is thus

$$O(|AN|^2\cdot(1+\bar{b})) \text{ key table entries.}$$

If $\bar{b}$ is $O(1)$, the total message space complexity for bootstrapping the entire network becomes $O(|AN|^2)$. This is a reasonable assumption assuming uniform node density, irrespective of the size of the AN. The number of neighbours of any one node can then be expected to be a function of the transmitting radius.

6 Discussion

The proposed protocol establishes an authenticated communication channel between nodes in a wireless ad-hoc network. It also provides nodes with reliable identifiers that are resistant to impersonation attacks. The impersonation attacks are thwarted by the fact that the binding between the public key and the identity of the node can be easily verified by any member of the network. To effectively impersonate a node would require access to its secret private key used for signing the messages. Note that the existence of an authenticated channel allows the creation of a secret channel, thus making our protocol a basis for a solution immune to eavesdropping attacks. Observe also that the use of public key signatures results in one more valuable property, i.e., non-repudiation.

We now consider the identity model assumed. It is possible that two nodes will generate the same public key. However, the probability of such an event is extremely small, i.e. $1/2^{1536}$, assuming a 1536-bit public key (recall that the size given in Table 2 is for the compressed public key, used in transmission). Furthermore, even if public keys are different, a collision in the hash function is still possible. However – by the birthday paradox – if

we assume 32-bit identifiers, then there would have to be an average of $1.2 * 2^{16}$, i.e. about $7.86 * 10^4$ nodes in the network for the probability of collision to exceed 1/2. If we assume 64-bit identifiers, then about $5.15 * 10^9$ nodes are required for the probability of a collision to exceed 1/2.

The proposed protocol has the following features.

- As the identifiers provided by our protocol are immune to impersonation attacks, they can be used in a reputation management system.
- If a higher level of security is desired, a secret communication channel can be built between nodes in the *AN*. This allows for inter-*AN* messages to be sent in encrypted form.
- It is possible that two nodes can select the same identifiers, but chances of this are very small. Note that these extremely rare occurrences can be dealt with by introducing a new message `DENY` that is sent by a node in the *AN* when it detects a collision. The joiner node should generate a new key pair in case of a collision.
- One might simplify our protocol so that it does not result in security associations (known public keys) between all pairs. Then the approach of [8] for secure routing despite incomplete set of security associations can be used.

While there exist key agreement proposals for ad-hoc networks with lower message overhead and better space and computation requirements, none of them exhibit all the properties of our proposed protocol. To our knowledge, this is the first general method for establishing authenticated channels that satisfies the three requirements (R1) – (R3) listed in Section 1.3. In particular, this protocol is fully self-organized. In that respect, it differs from the family of identity-based cryptosystems, where a trusted third party is required to compute the private key in the setup phase. While the logic of the protocol may appear easy to follow, the significance is that it establishes a baseline protocol meeting requirements (R1) – (R3). It is general enough to be incorporated in various protocol settings. This proposal should be considered as a first humble step in the process of finding completely self-organized solutions to the problem of securing ad-hoc networks.

7 Conclusions

This paper made two contributions.

- First, it proposed a fully self-organized protocol that establishes an authenticated communication channel between nodes of a wireless ad-hoc network. The protocol does not rely on the existence of a Trusted Third Party, the nodes do not need to share a prior common context, and no out-of-band communication channel is required. The protocol is independent of the upper layer protocols, and in particular, it is not an extension to any existing routing protocol. The resulting authenticated channel can be further used to establish a secret communication channel between nodes.
- Second, the protocol also provides a secure identifier framework that is resilient to impersonation. The authentic identifiers provided by the protocol can be used to associate network (and upper) layer identifiers to prevent spoofing. They can also serve as a reliable basis for reputation management protocols.

The protocol can be extended to handle multiple independent authenticated networks. This would be useful in a scenario where more than one *ANs* need to share the same bandwidth. It also allows for one node to be the member of multiple *AN*s. This is useful from the application perspective and also facilitates the implementation of some network functions such as firewalling and tunneling between different *AN*s.

References

[1] K. Aberer and Z. Despotovic, "Managing trust in a peer-to-peer information system," in *10th International Conference on Information and Knowledge Management*, pp. 310-317, Atlanta, Georgia, USA, 2001.

[2] N. Arora and R. K. Shyamasundar, "UGSP: Secure key establishment protocol for ad-hoc networks," in *First International Conference on Distributed Computing and Internet Technology (ICDCIT)*, pp. 391-399, Bhubaneswar, India, 2004.

[3] D. Balfanz, D. Smetters, P. Stewart, and H. Wong, "Talking to strangers: Authentication in adhoc wireless networks," in *Symposium on Network and Distributed Systems Security (NDSS '02)*, San Diego, California, USA, 2002.

[4] S. Blake-Wilson and A. Menezes, "Authenticated Diffie-Hellman key agreement protocols," in *Selected Areas in Cryptography (SAC'98)*, pp. 339-361, Kingston, Ontario, Canada, 1998.

[5] R. B. Bobba, L. Eschenauer, V. Gligor, and W. Arbaugh, "Bootstrapping security associations for routing in mobile ad-hoc networks," in *GLOBECOM'03, IEEE Global Communications Conference*, pp. 1511-1515 , San Francisco, California, USA, 2003.

[6] S. Braynov and T. Sandholm, "Incentive compatible mechanism for trust revelation," in *1st International Conference on Autonomous Agents and Multiagent Systems*, pp. 310-311, Bologna, Italy, 2002.

[7] S. Capkun, L. Buttyán, and J.-P. Hubaux, "Self-organized public-key management for mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 1, pp. 52-64, 2003.

[8] S. Capkun and J. P. Hubaux, "BISS: building secure routing out of an incomplete set of secure associations," in *2nd ACM Workshop on Wireless Security (WiSe'03)*, pp. 21-29, San Diego, California, USA, 2003.

[9] S. Capkun, J.-P. Hubaux, and L. Buttyán, "Mobility helps security in ad hoc networks," in *4th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2003)*, pp. 46-56, Annapolis, Maryland, USA, 2003.

[10] C. Castelluccia and J. H. Yi, *DoS-resistant Self-keying Mobile Ad-hoc Networks*, Technical Report RR-5373, INRIA-Rhone-Alpes, Nov. 2004.

[11] H. Deng and D. P. Agrawal, "TIDS: threshold and identity-based security scheme for wireless ad hoc networks," *Ad Hoc Networks*, vol. 2, no. 3, pp. 291-307, 2004.

[12] W. Du, J. Deng, Y. S. Han, and P. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," in *10th ACM Conference on Computer and Communications Security (CCS)*, pp. 42-51, Washington, DC, USA, 2003.

[13] W. Du, J. Deng, Y. S. Han, S. Chen, and P. K. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge," in *INFOCOM*, pp. 586-597, 2004.

[14] W. Du, J. Deng, Y. S. Han, P. Varshney, J. Katz, and A. Khalili, "A pairwise key pre-distribution scheme for wireless sensor networks," *ACM Transactions on Information and System Security*, vol. 8, no. 2, pp. 228-258, 2005.

[15] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *ACM Conference on Computer and Communications Security*, pp. 41-47, 2002.

[16] M. Gupta, P. Judge, and M. Ammar, "A reputation system for peer-to-peer networks," in *13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 144-152, Monterey, California, USA, 2003.

[17] J.-P. Hubaux, L. Buttyán, and S. Capkun, "The quest for security in mobile ad hoc networks," in *2nd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, pp. 146-155, Long Beach, CA, USA, 2001.

[18] A. Khalili, J. Katz, and W. A. Arbaugh, "Toward secure key distribution in truly ad-hoc networks," in *IEEE Workshop on Security and Assurance in Ad-Hoc Networks*, pp. 342-346, 2003.

[19] D. Liu, P. Ning, and R. Li, "Establishing pairwise keys in distributed sensor networks," *ACM Transactions on Information and System Security*, vol. 8, no. 1, pp. 41-77, 2005.

[20] G. Montenegro and C. Castelluccia, "Crypto-based identifiers (CBIDs): Concepts and applications," *ACM Transactions on Information and System Security*, vol. 7, no. 1, pp. 97-127, 2004.

[21] G. O'Shea and M. Roe, "Child-proof authentication for MIPv6 (CAM)," *SIGCOMM Computer Communication Review*, vol. 31, no. 2, pp. 4-8, 2001.

[22] B. Sieka and A. D. Kshemkalyani, "Fully self-organized key agreement for ad-hoc wireless networks," in *IEEE Consumer Communications and Networking Conference (CCNC 2006)*, pp. 80-85, Las Vegas, NV, USA, 2006.

[23] F. Stajano and R. J. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks," in *7th International Workshop on Security Protocols*, pp. 172-194, Cambridge, UK, 1999.

[24] B. Yu and M. Singh, "An evidential model of distributed reputation management," in *1st International Conference on Autonomous Agents and Multiagent Systems*, pp. 294-301, Bologna, Italy, 2002.

[25] Q. Zhang, P. Wang, D. S. Reeves, and P. Ning, "Defending against sybil attacks in sensor networks," in *Second International Workshop on Security in Distributed Computing Systems (SDCS)*, pp. 185-191, 2005.

Bartlomiej Sieka has founded Semihalf, a software development company specialising in embedded systems and computer security. He received a Ph.D. in Computer Science from the University of Illinois at Chicago in 2006. He received a M.S. in Computer Science from the Jagiellonian University in Cracow, Poland in 1999 and a M.S. in Computer Science from the University of Illinois at Chicago, USA, also in 1999. His research interests include computer network security with particular emphasis on wireless, ad-hoc, and peer-to-peer networks.

Ajay Kshemkalyani is an Associate Professor at the University of Illinois at Chicago since 2000, before which he spent several years at IBM Research Triangle Park working on various aspects of computer networks. He received a Ph.D. in Computer Science from The Ohio State University in 1991, and a B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 1987. His research interests are in computer networks, distributed computing, and algorithms.