Entity Recognition for Sensor Network Motes (Extended Abstract)

Stefan Lucks[1], Erik Zenner[2], André Weimerskirch[3], Dirk Westhoff[4]

[1] Theoretische Informatik, University of Mannheim, Germany
[2] Erik Zenner, Cryptico A/S, Copenhagen, Danmark
[3] escrypt - Embedded Security GmbH, Bochum, Germany
[4] NEC Heidelberg, Germany

Abstract: Message authenticity (knowing "who sent this message") is an important security issue for sensor networks, and often difficult to solve. Sometimes, it may be sufficient and more efficient to solve the simpler *entitiy recognition* problem, instead: "is the message from the same entity that sent the previous messages?". This paper describes entity recognition for sensor network motes. A protocol presented at SAC 2003 [5] is shown to be insecure, and a new and provably secure protocol is proposed.

1 Introduction

Consider the following story: Two strangers, Alice and Bob, meet at a party and make a bet. Days later, after it had turned out that Alice is the winner, Bob receives a message: *"Bob, please transfer the prize to bank account [. . .] Thank you. Alice."*. How does Bob know that this message actually has been sent from that person, who had called herself "Alice" at that party? In other words, how does Bob recognise a message from Alice?

In this paper, Alice and Bob are sensor network motes. Using digital signatures would be computationally expensive for them. We present a low-cost solution based on secret-key cryptography (cryptographic hashing and message authentication). We neither assume the existence of a trusted third party, nor the availability of pre-deployed secret or authentic information, the network topology can be dynamic, and there may be no (securely) synchronised time. Sensor networks typically have most or even all of these properties.

Scenario Description In short, we assume Eve, the adversary, to have *full control over the connection between Alice and Bob*. We consider this to be reasonably pessimistic: Over-estimating the adversary is not as bad as under-estimating her capabilities. Thus, Eve can

- ○ read all messages sent from Alice or from Bob,
- ○ modify messages, delay them or send them multiple times to either party,
- ○ and send messages generated by herself to Alice or Bob or both.

We have to make one exception, though. Without some faithfully relayed initial messages, the entire notion of "recognition protocols" would not make sense. Thus, we assume an **initial phase** (typically with one message from Alice to Bob, and a second message from Bob to Alice), where Eve reads the messages, but she relays them faithfully.

Driven by reasonable pessimism as before, we assume that Eve aims for an *existential forgery* in a *chosen message* scenario: She can choose messages x_i for Alice to authenticate and send ("commit"), and she succeeds if Bob accepts any message $x' \neq x_i$ as authentic.

More formally, we write commit-message(x_i,i) if Alice authenticates and sends the message x_i in time-frame i. In practice, x_i will be a value from outside the scope of the protocol, e.g., a sensor measurement. It should be anticipated that Eve has some influence on x_i, and in theory, we assume that Eve can choose x_i. We write accept-message(x_i, i), if Bob believes the message x_i to be authentic and fresh in time-frame i.[1] Eve wins if she somehow can make Alice to commit-message(x_i,i) and Bob to accept-message(x', i).

Since Eve has full control over the connection between Alice and Bob, the reliability of the connection depends on her[2]. Thus, *denial of service* attacks are trivial for Eve. We point out, however, that our solution is *sound* (i.e., if Eve works like a passive wire, the protocol works as intended) and supports *recoverability*: if, after some suppressed or modified messages, Eve again begins to honestly transmit all messages, like a passive wire, the soundness with respect to new messages is regained.

Security Parameters and Cryptographic Base Operations Let c and s be security parameters. We consider s to be the size of a symmetric key and c to be the output size of a message authentication code. In a typical application scenario, we would require $s \geq 80$ and $c \geq 30$. The two building blocks in this paper are a cryptographic hash function h (which we actually use as a one-way function $h : \{0,1\}^s \rightarrow \{0,1\}^s$), and a message authentication code (in short: a "MAC") $m : \{0,1\}^s \times \{0,1\}^* \rightarrow \{0,1\}^c$. Hash functions and MACs are rather cheap to implement and evaluate. We write $x \in_{\mathrm{R}} \{0,1\}^s$ for the uniformly distributed random choice of a hash input. Finally, n is a predefined constant (the maximal number of messages to authenticate).

2 Attacking a Proposed Solution

In [5] an entity recognition protocol is proposed, and a proof of security is presented. Unfortunately, the proof is flawed, and an attack against that protocol can be given.[3]

At first, Alice chooses a random value a_0 and generates a hash chain $a_1 := h(a_0), \ldots, a_n := h(a_{n-1})$. Similarly, Bob chooses b_0 and generates $b_1 := h(b_0), \ldots, b_n := h(b_{n-1})$.

[1]Our notion of freshness implies some (small) time frame for each x_i, which is known to Bob. The message x_i is "fresh" in frame i, if Alice had actually committed to x_i within frame i. During a time frame, Alice only commits to one single message, and Bob accepts (at most) one such message.

[2]In practice, there can also be non-hostile reasons for a connection to become unreliable.

[3]The proof in [5] implicitly assumes either party to notice when the other party rejects a message. In communication scenarios relevant for entity recognition, this is hardly realistic.

The **initial phase** consists of two messages: Alice $\to$ Bob: a_n, Bob $\to$ Alice: b_n. After the initial phase, Alice's internal state can be described by the triple (b_n, n, 1), and Bob's by (a_n, n, 1). During protocol execution, we write (b_i, j, u) for Alice's internal state and (a_j, i, v) for Bob's[4]. **Authenticating a text** x goes like this:

1. Alice $\to$ Bob: $m(a_{j-u-1}, x)$, a_{j-1}.
2. Bob verifies $h(a_{j-1}) = a_j$.
3. For k := 1 to k' := max$\{u, v\}$ do
   a) Bob $\to$ Alice: b_{i-k}.
   b) Alice verifies $h(b_{i-k}) = b_{i-k+1}$.
   c) Alice $\to$ Bob: a_{j-k-1}.
   d) Bob verifies $h(a_{j-k-1}) = a_{j-k}$.
   e) If any verification fails or the loop is interrupted,
   then (Alice and Bob abort) Alice's new internal state is (b_i, j, max$\{u, k+1\}$).
   Bob's new internal state is (a_j, i, max$\{v, k+1\}$).
   Else (both continue)...... Alice's new internal state is ($b_{i-k'}$, $j-k'-1$, 1).
   Bob's new internal state is ($a_{j-k'}$, $i-k'-1$, 1).

Let Alice's internal state be $(b_i, j, 1)$ and Bob's $(a_j, i, 1)$. The **attack** works as follows:

1. Alice $\to$ Bob: $m(a_{j-2}, x)$, a_{j-1}.
2. Bob verifies $h(a_{j-1}) = a_j$. **(OK!)**
3. For k := 1 to 1 do
   a) Bob $\to$ Alice: b_{i-1}.
   b) Alice verifies $h(b_{i-1}) = b_i$. **(OK!)**
   c) Alice $\to$ Bob: a_{j-2}. **Manipulation:** Eve changes a_{j-2} to $a' \neq a_{j-2}$.
   d) Bob verifies $h(a') = a_{j-2}$. **(Check fails!)**

Thus, Alice sends a_{j-2} in Step 3.c, but Bob receives $a' \neq a_{j-2}$. Since Alice's check is OK, her internal state becomes (b_{i-1}, $j-2$, 1). On the other hand, Bob's check fails, thus his new internal state is (a_i, j, 2). Now assume the next message x' to authenticate:

1'. Alice $\to$ Bob: $m(a_{j-4}, x')$, a_{j-3}.
2'. Bob verifies $h(a_{j-3}) = a_j$. **(Check fails!)**

At first glance, this is a denial of service attack: Eve modifies a single message, and the protocol stalls, because it lacks of *recoverability*[5]. But Eve can even *forge* any message x'': To accept x'', Bob needs to see a_{j-1}, a_{j-2}, and a_{j-3}, verifying $h(a_{j-1}) = a_j$, $h(a_{j-2}) = a_{j-1}$, and $h(a_{j-3}) = a_{j-2}$. In step 1', Alice sends a_{j-3} to Bob. *Eve, having seen a_{j-3}, can impersonate Alice and convince Bob to accept any x'' of Eve's choice.*

[4] The first value is the currently verified "endpoint" of the other party's hash chain, the second points into the own hash chain, and the third counts the number of necessary repetitions.

[5] In fact, any random corruption of a_{j-2} is likely to break the service.

3 A Description of our Protocol

In this section, we describe a new protocol to solve the entity recognition problem without using public-key cryptography. For initialisation, Alice chooses a random value a_0 and generates a hash chain $a_1 := h(a_0), \ldots, a_n := h(a_{n-1})$. Similarly, Bob chooses b_0 and generates $b_1 := h(b_0), \ldots, b_n := h(b_{n-1})$. When running the protocol, both Alice and Bob learn some values b_i resp. a_i from the other's hash chain. If Alice accepts b_i as authentic, we write accept-key(b_i). Similarly for Bob and accept-key(a_i).

The **initial phase** consists of two messages: Alice → Bob: a_n, Bob → Alice: b_n. Thus, we have accept-key(a_n) and accept-key(b_n).

After that, we split the protocol into n epochs. The epochs are denoted by $n-1, \ldots, 0$ (in that order). Each epoch allows Alice to send a single authenticated message, and Bob to receive and verify it. The internal state of each Alice and Bob consists of an epoch counter i, the most recent value from the other's hash chain (accept-key(b_{i+1}) for Alice and accept-key(a_{i+1}) for Bob), and a one-bit flag to select between program states A0 and A1 for Alice resp. B0 and B1 for Bob. Also, both Alice and Bob store the root a_0 resp. b_0 of their own hash chain.[6] This value doesn't change during the execution of the protocol.

After the initial phase, and before the first epoch $n-1$, Alice's state is $i = n-1$, accept-key(b_n), and A0, and Bob's is $i = n-1$, accept-key(a_n), and B0. **One epoch** i can be described as follows:

A0 *(Alice's initial state)* Wait for x_i *(from the outside)*, then continue:
  commit-message(x_i,i); compute $d_i = m(a_i, x_i)$ *(using a_i as the key to authenticate x_i)*;
  send (d_i, x_i); **goto** A1.
A1 Wait for a message b' *(supposedly from Bob)*, then continue:
  if $h(b') = b_{i+1}$ **then** $b_i := b'$; accept-key(b_i); send a_i; set $i := i-1$; **goto** A0;
  else goto A1.
B0 *(Bob's initial state)* Wait for a message (d_i, x_i), then continue: send b_i and **goto** B1.
B1 Wait for a message a' *(supposedly from Alice)*, then continue:
  if $h(a') = a_{i+1}$
  then $a_i := a'$; accept-key(a_i);
    if $m(a', x_i) = d_i$ **then** accept-message(x_i, i); *(authentic in epoch i)*
    set $i := i-1$; **goto** B0;
  else goto B1.

If, in state B1, Bob is sent a' with $h(a') = a_{i+1}$ but $m(a', x_i) \neq d_i$, Bob will set $i := i-1$; and go to state B0. Accordingly, no message will be accepted as "authentic in that epoch".

One epoch consists of two messages from Alice to Bob and one from Bob to Alice, see Fig. 1. The protocol is **sound**: If all messages are faithfully relayed, Alice commits to x_i in the beginning of epoch i and Bob accepts x_i at the end of the same epoch. Also, the protocol can **recover** from message corruption: Repeating old messages can't harm security, Eve may know them, anyway. We thus allow Alice to re-send a_{i+1} and (x_i, d_i), if she is in

[6] Storing a_0 and b_0 is sufficient to derive all other hash values, but for improved performance, Alice and Bob can implement a time-storage trade-off [2].

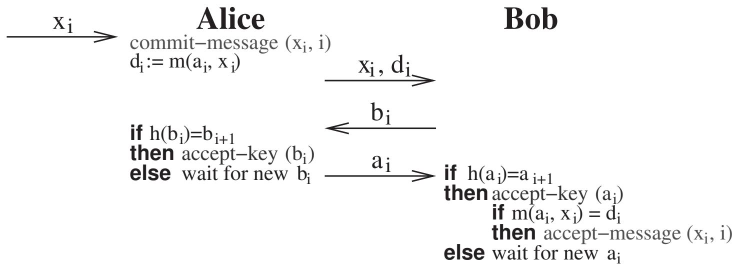

Abbildung 1: Simplified description of one epoch of the protocol

state A1 and has been waiting too long for the value b_i from Bob. Similarly, if Bob is in state B1 and has been waiting too long for a_i, Bob sends the value b_i again.

4 Final Comments

As we demonstrated by the *attack* in Section 2, designing secure entity recognition protocols is tricky and error-prone. Thus, it is desirable to prove the security of a proposed protocol. Further, we argue that authentication and recognition protocols for sensor networks must be rather efficient to run on extremely low-cost devices and to save energy resources [6]. Our protocol (a) is extremely efficient, (b) does not need a trusted third party or a key pre-distribution scheme, and (c) is provably secure. In the full paper [4], we **formally prove the security of our protocol**, assuming the security of the primitive operations h and m.

Literatur

[1] P. Buondonna, J. Hill, D. Culler. "Active Message Communication for Tiny Networked Sensors".

[2] D. Coppersmith and M. Jakobsson. "Almost Optimal Hash Sequence Traversal". Financial Cryptography 2002.

[3] A. Hodjat, I. Verbauwhede. "The Energy Cost of Secrets in Ad-hoc Networks (Short Paper)". (2002). `citeseer.ist.psu.edu/hodjat02energy.html`

[4] S. Lucks, E. Zenner, A. Weimerskirch, D. Westhoff, "Is this a Message from Alice?" Submitted.

[5] A. Weimerskirch, D. Westhoff. "Zero Common-Knowledge Authentication for Pervasive Networks". SAC 2003.

[6] A. Weimerskirch, D. Westhoff, S. Lucks, E. Zenner, "Efficient Pairwise Authentication Protocols for Sensor Networks: Theory and Performance Analysis". Jennifer Carruth, Thomas F. La Porta (eds), *"Sensor Network Operations"*. IEEE Press Monograph, 2004.