Concrete Security for Entity Recognition: The Jane Doe Protocol

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

[1] Bauhaus-Universität Weimar, Germany ⟨http://medsec.medien.uni-weimar.de/⟩
[2] Technical University of Denmark ⟨http://www.erikzenner.name/⟩
[3] escrypt Inc., USA ⟨http://weimerskirch.org/⟩
[4] NEC Europe Ltd ⟨Dirk.Westhoff@nw.neclab.eu⟩

Abstract. Entity recognition does not ask whether the message is from some entity X, just whether a message is from the same entity as a previous message. This turns turns out to be very useful for low-end devices. The current paper proposes a new protocol – the "Jane Doe Protocol" –, and provides a formal proof of its concrete security. The protocol neither employs asymmetric cryptography, nor a trusted third party, nor any key pre-distribution. It is suitable for light-weight cryptographic devices such as sensor network motes and RFID tags.

1 Introduction

Consider the following story: Two strangers meet at a party and make a bet. They introduce themselves as Jane and John Doe, which may or may not be their real names. Some days later, however, it turns out that Jane is the winner, and John receives a message: *"John, please transfer the prize to bank account [...] Thank you. Jane."*. How does John know that this message actually has been sent from that person, who had called herself "Jane" at that party? In other words, how does John recognise Jane – or a message from her?

Below, we will use the names Alice and Bob instead of Jane and John Doe for sender and receiver. As the protocol goal is about entity recognition, "real" names are unimportant. Alice and Bob are technical devices communicating in a hostile environment. Recognising each other would be easy if they could use unique identities and digital signatures: Initially, Alice would send Bob her public key. Later, Alice would sign all the messages she sends to Bob, and Bob would verify these signatures. But digital signatures are computationally expensive, and may seem an "overkill" to the problem at hand.

In this paper, we present the **Jane Doe protocol**, a light-weight solution to entity recognition using only symmetric primitives (namely, message authentication codes). Even low-end devices, which are too slow for digital signatures or the like, can run our protocol. The protocol does not depend on any trusted third party. Neither does it require a pre-established common secret key. It runs efficiently enough for real-time applications. In addition, it is interactive and provides information about the freshness and timeliness of messages.

Our research is motivated by the emergence of extremely low-power and low-cost devices such as sensor network motes and RFID tags. The continued desire to make these devices smaller at an attractive price offsets the technological advancements of increasing computational power. While implementing digital signatures and public-key techniques on such devices is technologically feasible, it is a hard burden from an economic viewpoint. Also, such devices are often used in networks where one can neither assume availability of a trusted third party, nor availability of pre-deployed secret or authentic information, and with a dynamic network topology. Another motivation is the question to what degree one can imitate the functionality of public-key cryptography and digital signatures by just using some simple primitives from symmetric cryptography. The Jane Doe protocol turns out to be as powerful as the common two step protocol for authenticating messages, consisting of a non-authenticated Diffie-Hellman key agreement at initialisation time followed by MAC authenticated messages.

Previous Work: The security goal of entity recognition has independently been proposed by a couple of different authors under different names [2, 18, 16, 10, 8].

An early protocol to actually address entity recognition was the **Resurrecting Duckling protocol** [17]. As it requires the exchange of a secret key in the initialisation phase, it does meet our security requirements. The **Guy Fawkes protocol** by [1] is more suitable for entity recognition, but it implicitly assumes Alice to know when Bob has seen her commitment a_i. While this may be the case in the original use case (Guy Fawkes would publish his commitments in a newspaper), an explicit confirmation of receipt may be desirable in most application contexts. The **Remote User Authentication protocol** [14] uses a message authentication code (MAC) and a cut-and-choose approach, which is much more demanding than our protocol. In [15], messages are authenticated using MACs, with a symmetric key being exchanged **using Diffie-Hellman key exchange** at protocol start. The problem here is that the key exchange requires public-key operations, which are too onerous for low-end systems. In the full paper [13], we provide a rough comparison of this approach with our proposal. The **zero-common-knowledge protocol** [18] from SAC 2003 uses hash chains, like our protocol, but turned out to be flawed [12, 13].

2 Scenario Description

Sending messages: Alice is the sender of messages, Bob the receiver. All protocols start with an **initialisation phase**, where Alice and Bob for the first time contact each other and exchange some initial material. Later, messages are sent from Alice to Bob in distinct time frames, which we denote as **epochs**. There can be at most n such epochs. Each such epoch i consists of four basic steps:

1. Alice receives some external data x_i, the origin of which lies outside the scope of the protocol (e.g. a measurement from a sensor).
2. Alice authenticates and sends the message to Bob. Formally, we write *CommitMessage*(x_i,i).

3. Bob sends a confirmation that he received some data, supposedly from Alice.
4. Alice opens the commitment and proves that it was really her who send the message. We write *AcceptMessage*(x_i, i) if Bob believes the message x_i to be authentic and fresh in epoch i.

Adversary capabilities: The well-known Dolev-Yao model [7] assumes that Eve is in full control over the connection between Alice and Bob, i.e. she is an **active adversary**. In particular, she can

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

This is considered as reasonable pessimism: Over-estimating the adversary is not as bad as under-estimating her capabilities. However, e.g. Gollmann [9] argues that novel applications may need more specific models. In our case, we make the special assumption that during the initialisation phase, Eve behaves like a **passive adversary**. She can read the messages between Alice and Bob (which precludes any kind of secret key exchange), but she relays them faithfully. Note that this is a weakening of the usual assumption that Alice and Bob can use a protected communication channel for initialisation, i.e. our scenario requires less external protection than most other proposals.

In typical application scenarios, Eve may even be able to extract secret data inside the devices by tampering, in addition to controlling the network. Our protocol does not protect against this kind of threat. If this threat is relevant for the application at hand, and if it can not be mitigated by using tamper-resistant hardware, then additional protection measures (like introducing redundancy and using secure multi-party computation algorithms) have to be introduced.

Adversary goal: Driven by reasonable pessimism as before, we assume that Eve aims for an *existential forgery* in a *chosen message* scenario:

- Eve may have some influence on x_i. Thus, for purposes of security analysis, we allow her to choose messages x_i which Alice will authenticate and send, i.e. *CommitMessage*(x_i,i).
- She succeeds if Bob accepts any message $x' \neq x_i$ as authentic, i.e. *AcceptMessage*(x', i).

At the beginning of the protocol, Alice and Bob choose initial random values a_0 resp. b_0. From then on, Alice and Bob act as strictly deterministic machines. When receiving a message, Alice and Bob update their internal state and send a response, if necessary. Eve is a probabilistic machine with independent connections to Alice and to Bob. In the context of this paper, the actual choice of a machine model is not important – any reasonable machine model will do.

We require the initial random values (=keys) a_0 and b_0 to be chosen independently from the keys for other sessions. To this regard, our setting is much simpler than any communication scenario where *the same* key material can be used in more than one session (see e.g. [4, 3]).

Limitation: We assume that the number of messages to be authenticated is known in advance, or a reasonable upper bound is known. During the initialisation phase, both Alice and Bob commit to the endpoint of a hash chain. The length of this hash chain bounds the number of messages to be authenticated. This limits of our approach, compared to other solutions employing public-key cryptography. Those, however, may be less efficient than our scheme, [13].

Reliability: Since Eve has full control over the connection between Alice and Bob, *denial of service* attacks are trivial for Eve. In addition, if the communication channel itself is unreliable, messages may be lost or faulty messages may be received even without the active involvement of a malicious adversary. Such problems can not be solved at cryptographical level, but have to be managed outside of the protocol. But the following reliability properties can be guaranteed:

Soundness: If the network is reliable and Eve behaves like a passive wire, the protocol works well: Bob accepts each message x_i Alice has committed to.

Recoverability: If Eve suppresses or modifies some messages, or creates some messages of her own, Bob may refuse to accept a message x_i Alice has committed to. However, once Eve begins again to honestly transmit all messages, like a passive wire, the soundness with respect to new messages is regained.

3 The Jane Doe Protocol

In this section, we describe the Jane Doe protocol to solve the entity recognition problem without using public-key cryptography. We write s for the size of a symmetric key. A second security parameter is the tag size $c \leq s$ for message authentication. (Typically: $s \geq 80$ and $c \geq 32$.) We use two functions, a MAC $m : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^c$ and a one-way function $h : \{0,1\}^s \to \{0,1\}^s$. (In Section 4, we will describe how to derive both m and h from a single MAC.) We write $x \in_{\mathrm{R}} \{0,1\}^s$ to indicate a random s-bit string x, uniformly distributed.

Initialisation phase: For initialisation, Alice chooses $a_0 \in_{\mathrm{R}} \{0,1\}^s$ and generates a hash chain $a_1 := h(a_0), \ldots, a_n := h(a_{n-1})$. Similarly, Bob chooses $b_0 \in_{\mathrm{R}} \{0,1\}^s$ 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 *AcceptKey*(b_i). Similarly for Bob and *AcceptKey*(a_i). The initialisation phase, where Eve can read the messages but relays them faithfully, consists of two messages:

1. Alice $\to$ Bob: a_n. (Thus: *AcceptKey*(a_n).)
2. Bob $\to$ Alice: b_n. (Thus: *AcceptKey*(b_n).)

Message authentication: We split the protocol up into n epochs, plus the initialisation phase. The epochs are denoted by $n-1, \ldots, 0$ (in that order). Each epoch allows Alice to send one authenticated message[1], and Bob to receive and verify it. The internal state of each Alice and Bob consists of

[1] Several messages can be sent per epoch. For ease of presentation, we combine them.

– an epoch counter i,
– the most recent value from the other's hash chain, i.e., b_{i+1} for Alice, and a_{i+1} for Bob (we write *AcceptKey*(b_{i+1}) and *AcceptKey*(a_{i+1})), 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.[2] This value does not change during the execution of the protocol. Note that after the initial phase, and before the first epoch $n-1$, Alice's state is $i = n-1$, *AcceptKey*(b_n), and A0, and Bob's is $i = n-1$, *AcceptKey*(a_n), and B0. One epoch i can be described as follows:

A0 (Alice's initial program state)
Wait for x_i (from the outside), then *CommitMessage*(x_i,i):
1. compute $d_i = m(a_i, x_i)$ (using a_i as the key to authenticate x_i);
2. send (d_i, x_i); **goto** A1.

A1 Wait for a message b' (supposedly from Bob), then
1. **if** $h(b') = b_{i+1}$
then $b_i := b'$; *AcceptKey*(b_i); send a_i; set $i := i-1$; **goto** A0
else goto A1.

B0 (Bob's initial program state)
Wait for a message (d', x') (supposedly from Alice), then
1. send b_i and **goto** B1.

B1 Wait for a message a' (supposedly from Alice), then
1. **if** $h(a') = a_{i+1}$ **then**
(a) $a_i := a'$; *AcceptKey*(a_i);
(b) **if** $m(a_i, x') = d'$
then $x_i := x'$; *AcceptMessage*(x_i,i)
(else do not accept any message in epoch i);
(c) set $i := i-1$; **goto** B0
else goto B1

Figure 1 gives a simplified view on the protocol.

Reliability: The following reliability properties are met:

Soundness: The protocol is **sound**: If all messages are faithfully relayed, Alice commits to the message x_i in the beginning of epoch i and Bob accepts x_i at the end of the same epoch.

Recoverability: Repeating old messages cannot 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 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. This allows our protocol to recover. On the other hand, if Bob receives a faulty $(x', d') \neq (x_i, d_i)$, he will refuse to accept *any* message in epoch i. Recovering means that soundness can be restored in epoch $i-1$.

[2] Alice can either store a_0 and compute the a_i on demand by making i calls to h, or store all the a_i using n units of memory. Her third option is to implement a *time-storage trade-off*, requiring only about $\log_2 n$ units of memory and $\log_2 \sqrt{n}$ calls to h [6]. Similarly for Bob and the b_i.

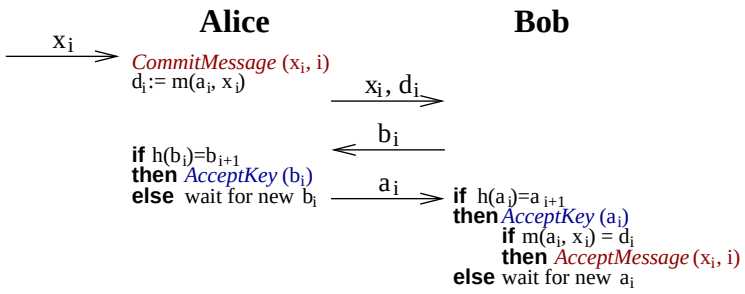


Fig. 1. Simplified description of one epoch of the protocol

4 Security

4.1 Building Blocks and Assumptions

The main cryptographic building block in this paper is a MAC

$$m^* : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^s$$

We fix some constant message const and define the two functions m and h we actually use in the protocol

$$h : \{0,1\}^s \to \{0,1\}^s, \quad h(k) = m^*(k, \text{const}), \quad \text{and}$$

$$m : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^c, \quad m(k,x) = \text{truncate-to-}c\text{-bit}(m^*(k,x)).$$

In the case of m, a restriction is $x \neq$ const. If neccessary, we we can, e.g., define const as a single zero-bit, and prepend a single one-bit to every message x.

Security against adaptive chosen message attacks has been established as a standard requirement for MACs:

Assumption 1 *It is infeasible for the adversary to provide an* ***existential forgery*** *in an* ***adaptive chosen message attack*** *scenario against m^*. I.e., the adversary is given access to an authentication oracle, computing $t_i = m(y, x_i)$ for the adversary, where $y \in_R \{0,1\}^s$ is secret and the adversary is allowed to choose arbitrary messages x_i. "Adaptive" means that the adversary is allowed to choose x_i after having seen t_{i-1}. The adversary wins if she can produce a pair (x', t') with $m(y, x') = t'$, without previously asking the oracle for $m(y, x')$.*

Unfortunately, this standard assumption is not quite sufficient for our purposes. Below, we will not make use of assumption 1 at all, but instead, define two similar assumptions. Firstly, we use m instead of m^* as a MAC, i.e., the truncation of m^* to $c \leq s$ bit. The security of m does not follow from the security of m^*. So we need to make the same assumption for m instead of m^*:

Assumption 2 *It is infeasible for the adversary to provide an* ***existential forgery*** *in an* ***adaptive chosen message attack*** *scenario against m.*

Furthermore, we use h to build a hash chain, which implies that h must be one-way. It may be surprising, but m^* being secure against existential forgery is not sufficient for the one-wayness of $h = m^*(\cdot, \text{const})$. If, given $k^* = h(k) = m^*(k, \text{const})$, the adversary can find the secret k, then she can forge messages. But the adversary could just as well find some value $k' \neq k$ with $k^* = h(k') = m^*(k', \text{const})$ without necessarily being able to to generate existential forgeries. We thus need to exclude this case:

Assumption 3 *The function m^* is **one-way**. I.e., given a random $k \in \{0,1\}^s$, and a message const, it is infeasible to find any $k' \in \{0,1\}^s$ with $m^*(k, const) = m^*(k', const)$.*

Note that inverting m^* (i.e., breaking the one-wayness of h) would either allow us to find a secret key and thus to forge messages, or provide a 2nd preimage, i.e., a value $k' \neq k$ with $h(k) = h(k')$. Indeed, for our formal proof of security we could replace assumption 3 by assuming 2nd preimage resistance. The proof would be slightly more complicated, though.

4.2 Proving Security for Epoch 0

Theorem 1. *If the adversary can efficiently break epoch 0 of the protocol, she can efficiently break either assumption 2 or assumption 3.*
Concrete security. If she can break the protocol in time t with probability p, she can either invert h or forge a message for m in time $\leq t + 2t^*$ with probability $p/2$. Here, t^* is the time to evaluate either h or m, which ultimately boils down to the time for evaluating m^*.

Proof. Eve can send the following messages (see also left side of Figure 2):

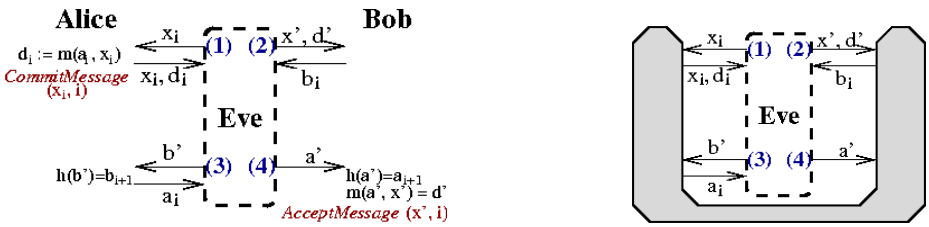
(1) If Alice's program state is A0: x_0 to Alice.
Alice responds $d_0 := m(a_0, x_0)$ (and x_0, but x_0 is known to Eve, anyway).
(2) If Bob's program state is B0: (x', d') to Bob – with $x' \neq x_0$.
(3) If Alice's program state is A1: b' to Alice – with $h(b') = b_1$.
(4) If Bob's program state is B1: a' to Bob – with $h(a') = a_1$.

Remember that she is successful if she gets Bob to *AcceptMessage*(x',i) for a message x' that Alice has not send in epoch i.

Note that (3)-like messages b' with $h(b') \neq b_1$ to Alice do not affect Alice's state; Alice ignores them. Since Eve can check $h(b') = b_1$ on her own, we assume w.l.o.g. Eve not to send any message b' with $h(b') \neq b_1$ to Alice. Similarly, for (4)-like messages, we assume, Eve not to send any a' with $h(a') \neq a_1$ to Bob.

In order to successfully attack, Eve *must* send *exactly* one message (1) to Alice (to ensure *CommitMessage*($x_0, 0$)) and both messages (2) and (4) to Bob (for *AcceptMessage*($x', 0$)). Eve may send *at most* one message (3) to Alice. W.l.o.g., we assume Eve to send *exactly* one message (3). (If she wins her attack game without sending message (3), she has sent message (2) and did learn b_0 from Bob. She can always send a final message (3) with $b' = b_0$.)

While (1,2,3,4) is the protocol-defined "natural" order for sending the messages, Eve is not bound to this order. There are some restrictions though:



Left: The four types of messages

Right: Eve, connected to some game

Fig. 2. Eve in epoch i

- Message (1) must be sent before message (3). Until she knows and has committed to x_0, Alice wouldn't even listen to message (3).
- Also, Bob wouldn't listen to (4) before having received (2).

In the context of this proof, we just need to distinguish between two cases, which we represent by two games: Either message (2) is sent before message (3), or the other way. Consider disconnecting Eve from Alice and Bob, and connecting her with either of two games (cf. right side of Figure 2). If we win such a game, we can either invert h or forge messages. We will show that Eve cannot distinguish her participation in such a game from the "real" attack against the protocol and show that a successful attack by Eve is essentially the same as us winning one of our games. So at the end, if Eve can feasibly attack the protocol, we can feasibly invert $h(\cdot) = m(\cdot, \text{const})$ or forge messages for m^*. The games are the following:

1st game (inverting h): Given $k^* = h(k) = m^*(k, 0)$, for a uniformly distributed random k, find some k' with $m^*(k', 0) = k^*$.

- Randomly choose a_0, compute $a_1 := h(a_0)$.
- If Eve sends message
  - (1), the value x_0: compute and respond $d_0 := m(a_0, x_0)$.
  - (2): abort the game.
  - (4): Report an error! (Message (2) must be sent before message (4), and this algorithm aborts after message (2).)
- When Eve sends (3), the value b': print $k' := b'$ and stop.

The values provided to Eve during the 1st game are distributed exactly as in the case of the real attack game. Namely, a_0 and b_0 are independent uniformly distributed random values, and all the other values are derived from a_0 and b_0. Note that if Eve sends message (3) before message (2), the game succeeds; else it doesn't. To compute a_1, we call h. To compute d_0, we call m. Thus, we need two function calls. As Eve herself runs in time t, the game takes time $t + 2t^*$.

2nd game (existential forgery for m): Consider an unknown random y, known $y^* = h(y)$, and the ability to ask an oracle for $m(y, \cdot)$. Proceed as follows.

- Set $a_1 := y^*$; randomly choose b_0; compute $b_1 := h(b_0)$.

– If Eve sends message
  - (1), the value x_0: ask the oracle for the response $d_0 = m(y, x_0)$.
  - (3): abort the game.
  - (4): Report an error!
– When Eve sends (2), the pair (x', d'): print (x', d') and stop.

Eve's attack succeeds if and only if (x', d') is an existential forgery.

Similarly to above, the distribution of values provided during the game is identical to the real attack game. The only computation during the game is the one for $b_1 := h(b_0)$, so the game needs time $t + t^* \leq t + 2t^*$.

Completing the proof: The 1st game is the counterpart of the second game: one succeeds if message (2) is sent before message (3), the other one, if message (3) is sent before message (2). Eve doesn't know which game we play – or rather, that we are playing games with her at all, instead of mounting the "real" attack. So Eve still succeeds with probability p. If we randomly choose the game we play, we succeed with $p/2$. Neither game takes more than time $t + 2t^*$. □

4.3 Security in any Epoch i

At a first look, it may seem that the security proof for epoch 0 is also valid for epochs $i > 0$. But in epoch 0, the keys for the MAC m^* are uniformly distributed random values a_0 and b_0 in $\{0,1\}^s$, while later, we use a_i and b_i:

– Our security assumptions for m^* require *uniformly distributed* random keys.
– Our security assumptions for m^* do not ensure the *uniform distribution* of the output values $a_i = h(a_{i-1}) = m^*(a_{i-1}, 0)$ and $b_i = \ldots$

Now m^* could be defined such that the one-way function $h(x) = m^*(x, 0)$ permutes over $\{0,1\}^s$. This would solve our problem, but restrict our choices m^* too much. In practice, however, most cryptographic MACs can reasonably be assumed to behave *pseudorandomly*. Thus, we make an additional assumption.

Let $u \in_{\mathrm{R}} \{0,1\}^s$ be a random variable chosen according to the uniform distribution. Let w be a random variable chosen by applying the function h to a uniformly distributed input, i.e., $v \in_{\mathrm{R}} \{0,1\}^s$, and $w := h(v)$. Let A be a distinguishing adversary for u and w. The advantage Adv_A of A in distinguishing u from w is defined in the usual way:

$$\mathrm{Adv}_A = \big|\Pr[A(u) = 1] - \Pr[A(w) = 1]\big|$$

Assumption 4 *No efficient adversary A can feasibly distinguish the distribution of the random variable $w = h(v)$, $v \in_{\mathrm{R}} \{0,1\}^s$, from the distribution of $u \in_{\mathrm{R}} \{0,1\}^s$. I.e., for all efficient A the advantage Adv_A is negligible.*

Recall that h is defined by $h(\cdot) = m^*(\cdot, \mathrm{const})$. For typical MACs m^*, this assumption is highly plausible.

We use assumption 4 to prove the pseudorandomness of values $a_1 := h(a_0)$, $\ldots$, $a_n := h(a_{n-1})$ for a random a_0, along an entire hash chain.

Lemma 1. *If, for any $i \in \{1, 2, \ldots, n-1\}$, the adversary can efficiently distinguish a_i from a_{i-1}, she can also distinguish a_1 from a_0, thus breaking Assumption 4.*
Concrete security. Let $i \in \{1, 2, \ldots, n-1\}$ be given. If the adversary can distinguish a_i from a_{i-1} in time t with an advantage α, she can distinguish a_1 from a_0 im time at most $t + (i-1) * t^*$ with the same advantage α. Here, t^* is the time for evaluating h.

Proof. Let a value r_0 be given, either distributed like a_0 or like a_1. Compute $r_1 := h(r_0) \ldots, r_{i-1} := h(r_{i-2})$. Now, r_{i-1} is either distributed like a_{i-1}, or like a_i, and we can distinguish between both options for r_{i-1} in the same time and with the same advantage as for a_{i-1} and a_i. Computing r_{i-1} takes at most $i-1$ calls to h. □

One more issue has to be taken into account. In the single-epoch case, we argued that finding 2nd preimages, i.e., values $a' \neq a_i$ with $h(a') = h(a_i) = a_{i+1}$ when given a_i, is infeasible under our assumptions. But when dealing with more than one epoch, Eve might possibly trick Alice into committing to some new message x_{i-1} and sending $d_i := m(a_{i-1}, x_{i-1})$ – even before Bob has seen a_i (see below). In contrast to an ordinary 2nd preimage attack, Eve now does not just know a_i, but she also has some additional information about a_{i-1}. Driven by the usual reasonable pessimism, we even assume Eve to know a_{i-1} itself. We consider finding an $a' \neq a_i$ with $h(a') = h(a_i) = a_{i+1}$ as a *guided 2nd preimage*. Theoretically, such guided 2nd preimages might be possible, even under all the assumptions we made so far. Thus, we make one additional assumption.

Assumption 5 *It is infeasible to find* guided 2nd preimages *for h. I.e., given $a_0 \in_R \{0,1\}^s$, $a_1 = h(a_0)$, and $a_2 = h(a_1)$, it is infeasible to find any $a' \neq a_1$ with $h(a') = a_2$.*

Recall that the adversary wins in epoch i if she can make Alice to *CommitMessage*(x_i,i) and Bob to *AcceptMessage*(x', i) for any $x' \neq x_i$.

Theorem 2. *If there is any epoch $i \in \{0, \ldots, n-1\}$ in which the adversary can feasibly win with significant probability, at least one of the assumptions 2, 3, 4, or 5 is false.*
Concrete security. If she can win in epoch i, in time t with probability p, she can either invert h, forge a message for m, or generate a guided 2nd preimage for h in time $\leq t + 2t^*$ with probability $p/4$. Or she can distinguish (a_i, b_i) from (a_{i-1}, b_{i-1}) with advantage $p/4$. Here, t^* is the time for calling either h or m, which ultimately boils down to calling m^*.

Proof. We say, the protocol in a "synchronised state", if there is an $i \in \{0, \ldots, n\}$ such that Bob knows a_i but not a_{i-1}, while Alice knows b_i but not b_{i-1}. I.e., the protocol is in a synchronised state if both Alice and Bob are in the same epoch $i-1$. After the initialisation, both are in epoch $n-1$, hence the protocol is in a synchronised state.

For the proof, we need to analyse independently how Eve can benefit from *non-synchronised states*, and how she can benefit from *synchronised states*.

Non-synchronised states: Consider Alice and Bob to be in epoch i, thus the protocol state is synchronised. Alice will not move forward into epoch $i-1$ without having seen b_i with $h(b_i) = b_{i+1}$. If Eve could provide such a b_i without obtaining it from Bob, she could win in epoch i anyway. Thus we can safely assume that Alice does not move forward before Bob sends b_i. For the same reason, we may assume Bob not moving forward to epoch $i-1$ without having seen a_i from Alice. Bob only sends b_i *after* having seen a_i from Alice. Thus, Bob can never be ahead of Alice. Temporarily, Alice can be ahead of Bob – especially if Eve does not forward a_i to Bob. This would give a protocol state with Alice living in epoch $i-1$ while Bob still lives in epoch i. But without having seen b_{i-1}, Alice cannot move ahead into epoch $i-2$, and Bob does not send this while he is still in epoch i.

At this point, Eve has but two options to proceed. One is to forward a_i to Bob, thus creating a new synchronised state. The second is to choose a message x_{i-1} and send it to Alice, who responds with the authentication tag $d_{i-1} = m(a_{i-2}, x_{i-1})$. If, after sending x_{i-1} to Alice, Eve sends the value a_i to Bob which she has seen before, there is no gain for Eve. The order of messages has changed, but the messages are the same, anyway. To benefit from the second option, Even has to send a value $a' \neq a_i$ with $h(a') = h(a_i) = a_{i+1}$ to Bob. If Eve could find such a value a', she could find guided 2nd preimages, thus breaking assumption 5.

Synchronised states: Now consider both Alice and Bob being in some epoch i, and Eve trying to win in this epoch. This part of the proof is done by induction. We start with epoch 0. Recall that if both assumption 2 and assumption 3 hold, the adversary cannot feasibly win in epoch 0.

Now assume that no efficient adversary can win in epoch epoch (i-1), but there is an efficient algorithm to win epoch i with significant probability. Clearly, we can use this algorithm to distinguish (a_{i-1}, b_{i-1}) from (a_{i-2}, b_{i-1}), thus breaking assumption 4.

Concrete security (sketch): This part is quite similar to the proof of theorem 1, the single-epoch case. Instead of two different games, we need to define four:

1. One game to invert h (like the 1st game in the proof of theorem 1).
2. One game to forge messages for m (like the 2nd game above).
3. One game to generate guided 2nd preimages for h.
4. One game to distinguish (a_{i-1}, b_{i-1}) from (a_{i-2}, b_{i-1}).

If Eve wins, we succeed in at least one of the games. Which game we succeed in depends on Eve's behaviour. As we must commit to one game in advance (i.e. before we know how Eve behaves), the probability of success decreases from p (for Eve) to $p/4$ (for us). □

5 Final Remarks and Conclusion

The Jane Doe protocol does not provide security against *denial of service attacks*. I.e., if Eve sends a fake d_i in epoch i, Bob will send b_i and then not accept the "real" d_i Alice may later send.

Freshness means that a message has been committed to recently. In our case, when Bob accepts message x_i in epoch i, he can be sure that Alice (following the protocol rules) did not commit to that message before she had seen and verified Bob's response b_{i+1} from the previous epoch. In this sense, our protocol ensures the freshness of the messages authenticated.

The messages are "fresh" by belonging to the current epoch. But Eve is able to stretch any epoch at her will. Assume, e.g., that Alice commits to a message $m_i =$"I am well", but Eve delays forwarding $d_i = m(a_i,$ "all is well") to Bob. Later, Alice would need to raise an alarm, but instead Eve forwards d_i to Bob who sends b_i, which Eve immediately forwards to Alice. The protocol logic would require Alice to reply a_i, thus confirming that she is well. Instead of confirming such an outdated message, Alice could simply terminate communication with Bob. Eve has the power to cut the communication between Alice and Bob, anyway, and Bob will eventually notice that Alice doesn't respond any more.

Assuming some underlying primitive (from which we derive m^*) *to behave like a random oracle* is theoretically sound and would allow us to greatly simplify our security proofs. But in practice, cryptographic primitives never behave like random oracles. Results in the random oracle model hardly provide any guideline for the choice of a good primitive. Our very specific standard model assumptions on m^* are meant to serve as such a guideline.

Note that we have two functions, a message authentication code (MAC) m and a hash function h, both of which are derived from another MAC m^*. In principle, one could choose m and h independently from each other, without deriving them from the same underlying primitive, as has been suggested in [12]. Under appropriate assumptions, one can still prove the security of the Jane Doe protocol. This requires more complex and less natural assumtions than those made here. Even if m is a secure MAC and h is modelled as a random oracle, the protocol may actually be insecure[13]. Deriving both m and h from one single primitive m^* thus saves us from some difficult technical issues.

Furthermore, we believe that deriving both h and m from the same underlying primitive is natural and meets practical necessities very well.

Conclusions: Entity recognition is an adopted security goal especially useful for constrained pervasive applications. The Jane Doe protocol provides entity recognition. The protocol is efficient, runs on on very low-end devices, and is provably secure. We believe this to be a significant step into the direction of provably secure protocols for low-end devices.

References

1. R. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. Needham, "A New Family of Authentication Protocols", *ACM Operating Systems Review*, vol. 32, 1998.
2. J. Arkko, P. Nikander, "Weak Authentication: How to Authenticate Unknown Principals without Trusted Parties", Proc. *Security Protocols Workshop 2002*.
3. M. Bellare, P. Rogaway, "Entity Authentication and Key Distribution", *Proc. Crypto 1993*, Springer LNCS vol. 773, 1994.
4. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, "Systematic design of two-party authentication protocols", *Proc. Crypto 1991*, Springer LNCS vol. 576, 1992.
5. P. Buonadonna, J. Hill, D. Culler, "Active Message Communication for Tiny Networked Sensors", *Proc. 20th Joint Conference of the IEEE Computer and Communications Societies*, IEEE, 2001.
6. D. Coppersmith, M. Jakobsson, " Almost Optimal Hash Sequence Traversal", *Proc. Financial Cryptography 2002*, Springer LNCS vol. 2357, 2004.
7. D. Dolev, A. Yao, "On the Security of Public Key Protocols", *IEEE Trans. Information Theory*, vol. 29(2), pp. 198-208, March 1983.
8. P. Dielsma, S. Mödersheim, L. Vigano, D. Basin, "Formalizing and Analyzing Sender Invariance", *Formal Aspects in Security and Trust* (FAST 2006).
9. D. Gollmann, "Protocol Design: Coming Down from the Cloud" (Invited Talk), *Workshop on RFID and Lightweight Crypto 2005*, available from: http://www.iaik.tugraz.at/research/krypto/events/
10. J. Hammell, A. Weimerskirch, J. Girao, and D. Westhoff, "Recognition in a Low-Power Environment", *Proc. ICDCSW 2005*, IEEE, 2005.
11. A. Hodjat, I. Verbauwhede. "The Energy Cost of Secrets in Ad-hoc Networks". *IEEE Circuits and Systems workshop on wireless communications and networking*, IEEE, 2002.
12. S. Lucks, E. Zenner, A. Weimerskirch and D. Westhoff "Entity Recognition for Sensor Network Motes", Vol. 2, Proc. INFORMATIK 2005, pp. 145–149, LNI Vol. P-68, ISBN 3-88579-379-0 (an early 5-page abstract of the current research).
13. S. Lucks, E. Zenner, A. Weimerskirch and D. Westhoff "Concrete Security for Entity Recognition: The Jane Doe Protocol (Full Paper)", eprint, full version of the current paper.
14. C. Mitchell, "Remote User Authentication Using Public Information", *Proc. Cryptography and Coding 2003*, Springer LNCS vol. 2898, 2003.
15. S. Russell, "Fast Checking of Individual Certificate Revocation on Small Systems", *Proc. 15th Annual Computer Security Application Conference*, IEEE, 1999.
16. J.-M. Seigneur, S. Farrell, C. Jensen, E. Gray, and Y. Chen, "End-to-end trust in pervasive computing starts with recognition", *Proc. SPC 2003*, Springer LNCS vol. 2802, 2004.
17. F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks", *Proc. Security Protocols 1999*, Springer LNCS vol. 1796, 1999.
18. A. Weimerskirch and D. Westhoff, "Zero Common-Knowledge Authentication for Pervasive Networks", *Proc. SAC 2003*, Springer LNCS vol. 3006, 2003.
19. A. Weimerskirch, D. Westhoff, S. Lucks, and E. Zenner, "Efficient Pairwise Authentication Protocols for Sensor and Ad-hoc Networks", *Sensor Network Operations*, IEEE Press, 2004.