On Message Recognition Protocols: Recoverability and Explicit Confirmation

Ian Goldberg
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario Canada N2L 3G1
Email: iang@cs.uwaterloo.ca

Atefeh Mashatan
School of Computer and Communication Sciences, EPFL
CH-1015 Lausanne, Switzerland
Email: atefeh.mashatan@epfl.ch

Douglas R. Stinson
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario Canada N2L 3G1
Email: dstinson@uwaterloo.ca

Abstract:

We look at message recognition protocols (MRPs) and prove that there is a one-to-one correspondence between stateless non-interactive MRPs and digital signature schemes.

Next, we examine the Jane Doe protocol and note its inability to recover in case of a certain adversarial disruption. We propose a variant of this protocol which is equipped with a resynchronization technique that allows users to resynchronize whenever they wish.

Moreover, we propose another protocol which self-recovers in case of an intrusion. This protocol incorporates the resynchronization technique within itself. Further, we enumerate all possible attacks against this protocol and show that none of the attacks can occur. Finally, we prove the security of the new protocol and its ability to self-recover once the disruption has stopped.

Finally, we propose an MRP which provides explicit confirmation to the sender on whether or not the message was accepted by the receiver.

Keywords: Cryptographic Protocols, Message Authentication, Message Recognition, Self-Recoverability, Explicit Confirmation, Pervasive Networks, Ad Hoc Networks

Biographical notes: Ian Goldberg is an Assistant Professor in the David R. Cheriton School of Computer Science at the University of Waterloo, and a member of that university's Cryptography, Security, and Privacy (CrySP) research group. His research currently focuses on developing usable and useful technologies to help Internet users maintain their security and privacy.

Atefeh Mashatan is a Postdoctoral Researcher in the the Security and Cryptography Laboratory (LASEC) at the Swiss Federal Institute of Technology, Lausanne. Her main research interests are lightweight cryptographic protocols and their provable security properties.

Douglas R. Stinson holds the position of Professor and University Research Chair in the David R. Cheriton School of Computer Science at the University of Waterloo. He is a member of the Cryptography, Security, and Privacy (CrySP) research group. Dr. Stinson's research interests include cryptography and computer security, combinatorics and coding theory, and applications of discrete mathematics in computer science.

1 Introduction

Message recognition in ad hoc networks has been motivated in the literature by the following example (Lucks, Zenner, Weimerskirch & Westhoff 2005). Consider Alice and Bob, two strangers who meet at a party for the first time. They make a bet before they leave the party. Later, the outcome turns out to be in favour of Alice, and a few days later, Bob receives a message claiming to be sent from Alice. The message includes a bank account number and asks Bob to deposit Alice's prize to that bank account. How can Bob be assured that this message was indeed sent from the entity who introduced herself as "Alice" at the party? That is, Bob wants to *recognize* "Alice", whoever she was, or a message that was sent from her. This problem has a solution if Alice and Bob exchange some information, which is not necessarily secret, at the party.

Alternatively, let Alice and Bob be two small devices in a hostile environment. They have previously "met" in an environment that allowed them to send authenticated messages, but the messages were not confidential. Later, Alice wants Bob to recognize her or recognize the messages sent from her to Bob. There is an adversary, Eve, who is trying to make Bob recognize Eve as Alice, or accept messages from Eve as sent from Alice, where Alice has never sent those messages. Note that we do not consider replay attacks as threats in this scenario; a simple message counter can address them. A message recognition protocol is considered to be secure if Eve's attempts are detected by Alice or Bob. This problem has been considered in a context where we are dealing with devices with low computational power which cannot handle public-key computations and where no pre-deployed shared secret exists. On the other hand, the devices have access to a narrow-band authenticated channel at the initialization step and are later placed in a constrained, possibly hostile, insecure environment.

Recall the following widely used definitions of entity authentication and message authentication from the Handbook of Applied Cryptography (Menezes, van Oorschot & Vanstone 1996). *Entity authentication* is a security notion which assures the identity of a participating party to a second party. *Message authentication*, on the other hand, provides data origin authentication with respect to the original message source and does not have to provide uniqueness and timeliness.

We now define *entity recognition*, a security notion related to the entity authentication. Entity recognition is a weaker security notion than entity authentication; entity recognition refers to the process where two parties meet initially and one party can be assured in future conversations that it is communicating with the same second party. It should also provide *uniqueness*, that is, the corroborative evidence obtained in this process should uniquely correspond to the identity of the claimant. It should also assure *timeliness*, that is, to provide verifiable evidence that the claimant is active at the time of, or immediately before, the evidence was obtained.

Message recognition is a weaker security notion than message authentication and it provides data integrity with respect to the data origin. It ensures that the entity who sent the message is the same in future conversations. However, it does not have to provide uniqueness or timeliness.

Public-key techniques such as digital signature schemes solve the problem of recognition easily. However, using these techniques in some scenarios may be very costly. For instance, there may be no pre-deployed authentic information accessible. Also, we may not be able to assume trusted third parties are available to form a trusted infrastructure. Further, we may be dealing with devices with very low computational power where public-key computations are too heavy to be carried out. On the other hand, secret-key techniques require the existence of a secure channel where the secret keys can be transmitted confidentially. In a dynamic environment with no infrastructure, this assumption may not be easily realized.

The question is which security objectives can be achieved among devices with low computational power in an environment where no pre-established authentic information exists and without the presence of a trusted third party. Weimerskirch and Westhoff (2003) argued that in such an environment, achieving authentication is not possible and that all one can achieve is recognition security. They further note that recognition is often all that is required in most dynamic environments. Hence, the weaker security goals of entity and message recognition are often pursued (Anderson, Bergadano, Crispo, Lee, Manifavas & Needham 1998, Mitchell 2003, Weimerskirch & Westhoff 2003, Hammell, Weimerskirch, Girao & Westhoff 2005, Lucks, Zenner, Weimerskirch & Westhoff 2008).

There are two communication channels considered in the setting of recognition protocols: an insecure broadband channel which is available all the time, and an authenticated non-confidential narrow-band channel, which is only accessible once, at the very beginning of the protocol. That is, the narrow-band channel is used for the initial session between two users and later sessions occur over the insecure channel.

The goal of the adversary Eve is to make Bob accept some message M as coming from Alice, where M was not in fact previously sent by Alice. The attack model consists of Eve giving various messages to Alice to send to Bob and interacting with the resulting protocol runs. Eve can observe (but not modify) the messages on the narrow-band channel, and has full access to observe, inject, change, or delete messages on the broadband channel. All of this can be done in an adaptive fashion. For the message recognition protocol to be considered secure, it must be difficult for any polynomially bounded adversary Eve to achieve her goal.

1.1 Our Contribution

We first prove that there is a one-to-one correspondence between stateless non-interactive MRPs and digital signature schemes. Digital signature schemes are well studied in the cryptographic literature and their computational complexity is usually much higher than the level achievable by the devices considered in our scenario of interest. Hence, the one-to-one correspondence indicates that stateless non-interactive MRPs are unsuitable for these devices, and prompts us to focus on either interactive MRPs or non-interactive stateful MRPs.

We next examine previous recognition protocols proposed in the literature (Anderson et al. 1998, Mitchell 2003, Weimerskirch & Westhoff 2003, Hammell et al. 2005, Lucks et al. 2008), and conclude that the Jane Doe message recognition protocol proposed by Lucks et al. (2008) is the most suitable. Hence, we look at this protocol in more detail and note that in case of a particular adversarial disruption, this protocol fails to recover. In other words, the adversary can trap one party in a state that he or she will no longer accept legitimate messages that were sent by the other party; this is much stronger than a denial of service attack, as even after the adversary completes the attack and has no further contact with the parties, the parties remain unable to communicate.

We propose two solutions to remedy this shortcoming. The first solution consists of a variant of the Jane Doe MRP which is equipped with a *separate* resynchronization technique that allows users to resynchronize whenever they wish or when they suspect an intrusion has occurred. This first solution is much simpler, in terms of its instructions, when compared to our second solution; however, this simplicity comes at the price of having a separate protocol, and having to decide when to call upon it.

It is, of course, also of interest to remedy the recoverability shortcoming without requiring a separate synchronization procedure. In other words, one would like to incorporate the resynchronization technique within the protocol itself. We refer to this property as *self-recoverability.* When a protocol exhibits self-recoverability, the parties involved do not have to negotiate when to resynchronize; this makes the whole system more robust. Our second solution is a new message recognition protocol, which is based on the Jane Doe MRP and it "self-recovers" in case of an intrusion and does not need a separate resynchronization process. Moreover, we analyze all possible attacks against our protocol and prove that they can succeed with only negligible probability. We also formally prove that our new protocol is secure and fully recovers once the disruptions have stopped.

In all previous MRPs, the claimant sends a message to the verifier and hopes that the message is accepted by the verifier. In particular, these protocols do not offer any explicit means for the claimant to learn if the verifier has accepted or rejected the message. We call this missing property *explicit confirmation.* This confirmation can be achieved, for example, by executing a session of MRP with the verifier sending a message to the claimant where the content of the message relays acceptance or rejection of the previous intended message. This solution, and other solutions of this kind, requires the participants to get involved in a procedure which has almost the same complexity as the original primitive. As our third protocol, we propose a new MRP which provides explicit confirmation without requiring any extra communication between the participants.

The rest of the paper is organized as follows. Section 2 is devoted to showing the one-to-one correspondence between stateless non-interactive message recognition protocols and digital signature schemes. Section 3 is dedicated to examining previous recognition protocols and noting their shortcomings. In Section 4, we describe our first solution to overcome the recoverability problem. Section 5 is devoted to our second solution which exhibits self-recoverability, while Section 6 is dedicated to proving the security of this proposal. We propose our MRP with explicit confirmation in Section 7 and we conclude the paper in Section 8.

2 Non-interactive MRPs

In this section, we prove that there is a one-to-one correspondence between stateless non-interactive message recognition protocols and digital signature schemes.

Since digital signature schemes have a higher computational complexity than we deem suitable for our low-power devices in the context of message recognition, this one-to-one correspondence discourages any further investigation of a stateless non-interactive MRP and convinces us to focus on interactive MRPs or non-interactive MRPs that incorporate state.

2.1 A General Stateless Non-Interactive MRP

A general non-interactive message recognition protocol, where all flows are going from Alice to Bob, consists of two flows. The first flow is the initialization step which happens only once. The second flow, occurring over the insecure channel, is sent once for each message to be authenticated. As a result, the message and any additional authenticating information are all being transmitted in one flow.

Figure 1 depicts a general stateless non-interactive message recognition protocol. The protocol is described in terms of three (possibly randomized) functions, denoted by *gen*, *compose*, and *decompose*. On input (1^k), where k is a security parameter, the function *gen* outputs a pair (a, b), and b is sent to Bob.

For each message Alice wishes to send to Bob, Alice uses *compose* to construct the message to be sent to Bob; *compose* has access to a. Similarly, for each message Bob receives from Alice, he calls *decompose* to process it while using b as an input as well.

It is important to note that any stateless non-interactive protocol in our network setting can be put in this form. For this protocol to be a secure MRP, it is required that $\mathit{decompose}(c', b) = \perp$ with overwhelming probability if $c' \neq \mathit{compose}(M, a)$ for some message M and a valid

Alice	Eve	Bob
compute $gen(1^k) = (a, b)$	$\xrightarrow{b}$	Receive b.

Alice	Eve	Bob
Input (M, Bob)		
$c \leftarrow compose(M, a)$	$\xrightarrow{c}$	Receive c'
		$d \leftarrow decompose(c', b)$ If $d = M'$, a valid message, output (M', $Alice$); otherwise, reject.

Figure 1: A General Stateless Non-interactive Message Recognition Protocol

pair (a, b) produced by *gen*. Moreover, it is required that $decompose(c, b) = M$ for any c output by $compose(M, a)$.

2.2 Digital Signature Schemes with Message Recovery

Next, we consider a particular class of digital signature schemes (DSS): those with *message recovery.* In section 2.3 we will use a simple transformation from an arbitrary DSS to one with message recovery in order to handle the general case.

Digital signature schemes with message recovery (DSSMR), Figure 2, are often formulated by three algorithms: key generation, sign and verify. The key generation algorithm $\mathcal{G}$ randomly produces a pair of public and private keys (PK, SK) for each signer. The signer uses SK to sign and PK is used by others to verify signatures. We assume that any receiver has access to the signer's public key PK via an authentic (but not confidential) channel. On input message M and a secret key SK, the signing algorithm $\mathcal{S}$, which may be randomized, outputs a signature s. On input public key PK and a signature s, the signature verifying algorithm, $\mathcal{V}$, either outputs M', a valid message, or it rejects s.

A signature s of M that is honestly computed using the secret key SK should be accepted by the verifying algorithm using the associated public key PK. In other words, for all M and all (PK, SK) generated by $\mathcal{G}$, it holds that $\mathcal{V}(PK, \mathcal{S}(M, SK)) = M$. We assume the standard attack model of adaptive chosen message attack, wherein Eve adaptively gives Alice various messages to sign. Eve's goal is to create a signature that Bob will accept as being Alice's signature on a message M that Alice, in fact, never signed. For the signature scheme to be considered secure, it must be difficult for any polynomially bounded adversary to achieve this goal.

2.3 Equivalence of Stateless Non-interactive MRPs and Signature Schemes

Given the aforementioned definitions and properties, we obtain the following result on the equivalence of digital signature schemes and stateless non-interactive message recognition protocols.

Theorem 1 *Given functions gen, compose, and decompose, any stateless non-interactive message recognition protocol can be transformed to a digital signature scheme (with message recovery). Conversely, any digital signature scheme (with or without message recovery) can be transformed to a stateless non-interactive message recognition protocol.*

The equivalence of the two schemes is straightforward: given *gen*, *compose*, and *decompose*, we need to construct $\mathcal{G}$, $\mathcal{S}$, and $\mathcal{V}$. Just let $\mathcal{G} = gen$, $\mathcal{S} = compose$, and $\mathcal{V} = decompose$.

The converse construction is well known and straightforward: if the digital signature scheme $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ has message recovery, then simply setting $gen = \mathcal{G}$, $compose = \mathcal{S}$, and $decompose = \mathcal{V}$ is sufficient. If the digital signature scheme does not have message recovery, the standard transformation $\mathcal{S}'(M, SK) = (M, \mathcal{S}(M, SK))$ (and the corresponding change to $\mathcal{V}$) yields one that does have message recovery, and we proceed as before.

Finally, we need to show that security is preserved under these transformations. This can be done easily using reductions, by showing that an adversary that can break the message recognition protocol can be used as a subroutine to break the signature scheme, and vice versa. The reductions are obvious, so we just give a brief sketch of the reduction in one direction. Suppose that an adversary Oscar can break the message recognition protocol. Now, suppose that an instance of the signature scheme is specified, with public key PK. We show how Eve can use Oscar as a subroutine to break the signature scheme. First, Eve gives $a = PK$ to Bob to set up an instance of the message recognition protocol. Now Oscar will adaptively choose various messages to be transmitted in the message recognition protocol. For each such message M, Eve gives M to Alice (in the signature scheme) and obtains the corresponding s. Eve then gives s to Oscar. (That is, Eve simulates Alice in the message recognition protocol using the Alice in the signature scheme.) Eventually, Oscar outputs a new c in the message recognition protocol that would be accepted by Bob (in the message recognition protocol). Eve defines $s = c$, which is a signature that will be accepted by Bob in the signature scheme.

Alice	Eve	Bob
compute $\mathcal{G}(1^k) = (SK, PK)$	$\xRightarrow{PK}$	Receive PK.

Alice	Eve	Bob
Input (M, Bob)		
$\mathcal{S}(M, SK) = s$	$\xrightarrow{s}$	Receive s'
		Compute $\tilde{M} = \mathcal{V}(s', PK)$ If $\tilde{M} = M'$, a valid message, output $(M', Alice)$; otherwise, reject.

Figure 2: A Digital Signature Scheme with Message Recovery

2.4 Stateful Non-Interactive MRPs and an Open Problem

Figure 3 depicts a general stateful non-interactive message recognition protocol. Here, α is the internal (persistent) state of Alice, initialized to a; β is the internal (persistent) state of Bob, initialized to b. The protocol uses three (possibly randomized) functions *gen*, *compose*, and *decompose* where *compose* has access to all of Alice's state, and can also update that state and, similarly, *decompose* has access to and can update Bob's state.

Following the ideas above, we are able to show that a 'receiver-stateless' non-interactive MRP, in which Bob does not have a state, but Alice has a state, can be transformed into a (stateful) DSS. However, the extra power that may be obtained by giving state to Bob is not clear; for example, if an adversary reorders messages, but does not otherwise alter them, a stateful Bob may notice this and potentially reject the messages, noting the interference, whereas a digital signature verifier would accept the reordered messages as valid. We leave the question of whether there exist receiver-stateful non-interactive MRPs more efficient than digital signature schemes as an open problem.

3 Previous MRPs

In this section, we review the existing protocols in the literature which provide entity or message recognition. The usability of each protocol is discussed in the context of networks with devices having low computation power and low communication bandwidth.

The Guy Fawkes protocol was proposed by Anderson et al. (1998). There are two variants of this protocol suggested and a one-way hash function is employed in both variants. In the first variant, random codewords, X_i, are chosen in each session and are refreshed each time a message, M_i, is authenticated. Alice commits to the message and the codewords and then publishes the commitment in a public directory which provides time-stamping services. Later, she reveals the committed values to prove that she is the same party who was involved in previous sessions. However, assuming the existence of a trusted party which provides time-stamping services is not realistic in most ad hoc network scenarios. The second variant does not require any interaction with a time-stamping provider and instead requires interaction of the authenticating party with the verifying party. The initialization phase of this protocol does not assume any authenticated channel; however, it requires digital signatures for authenticating the first blocks and codewords. This may not be suitable in ad hoc networks and, in particular, in low-power environments. Moreover, for a message to be authenticated in session i, users need to commit to it in the previous session. In the context of message recognition, this means that users are engaged in two sessions of this protocol to authenticate a single message, which may not be desirable.

An entity recognition protocol known as the Remote User Authentication Protocol was introduced by Mitchell (2003). In this protocol, a message authentication code (MAC) is used to prove that a user is the same entity involved in previous sessions. The protocol can be adapted to perform message recognition as well; however, this is not discussed in the paper. The setup phase of this protocol requires that t MAC values be sent over the authenticated channel. This may be costly since authenticated channels are usually of low bandwidth. Further, the "cut-and-choose" procedure in each round involves sending $2t$ MAC values and r secret keys. In order for the protocol to be secure, it is suggested that $t \geq 35$ and $r \approx t/2$. Hence, the amount of computation and communication here is large compared to other protocols that are providing entity or message recognition and it may not be suitable for settings with low power devices.

Weimerskirch & Westhoff (2003) introduced the Zero Common-Knowledge (ZCK) protocol. This protocol is the starting point of a series of recent publications; see for example (Hammell et al. 2005), (Lucks et al. 2005), (Lucks et al. 2008), (Mashatan & Stinson 2008). They use MACs and hash chains of the form $a_i = h(a_{i-1})$ and $b_i = h(b_{i-1})$, $i = 1, \ldots, n$, as keys for the MACs computed by Alice and Bob, respectively. Here, n is fixed at the beginning and h is a one-way hash function.

Hammell et al. (2005) implemented the ZCK protocol and provided measurements and observations as a proof of concept. They investigated whether the ZCK protocol suits devices with low computational power, low code space, low communication bandwidth, and low energy resources. They concluded that the protocol does meet these

Alice	Eve	Bob
compute $gen(1^k) = (a, b)$	$\xRightarrow{b}$	Receive b.
Initialize $\alpha = a$.		Initialize $\beta = b$.

Alice	Eve	Bob
Input (M, Bob)		
$(c, \alpha) \leftarrow compose(M, \alpha)$	$\xrightarrow{c}$	Receive c'
		$(d, \beta) \leftarrow decompose(c', \beta)$ If $d = M'$, a valid message, output $(M', Alice)$; otherwise, reject.

Figure 3: A General Stateful Non-interactive Message Recognition Protocol

requirements; however, denial-of-service and memory complexity were identified as areas of concern and needed to be addressed or improved upon in the future.

Note that Hammell et al. (2005) investigate the practicality of the ZCK protocol but do not investigate its security properties. That is, Hammell et al. rely on the security proof presented by Weimerskirch & Westhoff (2003). However, Lucks et al. found a flaw in the security proof of this protocol and presented an attack against the ZCK protocol. Moreover, using the same idea of using values in a hash chain as keys for MACs, they proposed a message recognition protocol that guards against the found attack. We describe the protocol proposed by Lucks et al. in more detail; it has been named the Jane Doe protocol (Lucks et al. 2008).

Mashatan & Stinson (2008) proposed a message recognition protocol which does not make use of hash chains. Since this protocol does not use the hash chaining technique, it no longer requires the small devices to save values of a hash chain in their memories; this relaxes the memory requirements and makes the protocol more suitable for ad hoc networks. However, this is achieved at the cost of having to send longer messages in each round.

In this paper, we focus of the protocols that make use of the hash chaining technique, the last of which is the Jane Doe protocol.

3.1 The Jane Doe Protocol

A one-way hash function $H : \{0,1\}^s \rightarrow \{0,1\}^s$ and a message authentication code MAC : $\{0,1\}^s \times \{0,1\}^* \rightarrow \{0,1\}^c$ are the building blocks of this protocol. Typical parameters are suggested to be $s \geq 80$ and $c \geq 30$. The maximum number of messages to be authenticated, or the maximum number of sessions, in the Jane Doe protocol is fixed to be n. Alice randomly chooses a_0 and forms a hash chain of the form $a_i = H(a_{i-1})$, $i = 1, \ldots, n$. Similarly, Bob randomly chooses b_0 and forms $b_i = H(b_{i-1})$, $i = 1, \ldots, n$. Alice and Bob will respectively use a_i and b_i as keys for MAC values they compute in session i.

The initialization phase is constituted of Alice and Bob exchanging the values of a_n and b_n. In this phase of the execution, Eve is passive and the communication is denoted by $\Rightarrow$. Figure 4 illustrates the initialization phase of the Jane Doe protocol.

There will be n sessions of the protocol and we denote them in descending order by $n-1, \ldots, 0$; this is because the values of the hash chains are going to be revealed in this order. In each session i, Alice would like to authenticate a message m_i. She uses a_i as the key for the MAC and sends the MAC value of m_i to Bob. Bob then authenticates himself to Alice by revealing b_i. Once Alice has verified b_i, she reveals a_i. Then a_i allows Bob to verify Alice and m_i. Once the session is over, Alice and Bob "move down" in the hash chain and use a_{i-1} and b_{i-1} as keys for session $i-1$.

Lucks et al. write accept-key(k) when a key k has been accepted, and commit-message(m, i) when Alice commits herself to authenticate m in session i. Similarly, accept-message(m, i) indicates that Bob has accepted m as sent from Alice in session i. The formal description of the Jane Doe protocol is given next.

Alice's internal state in the Jane Doe protocol is as follows:

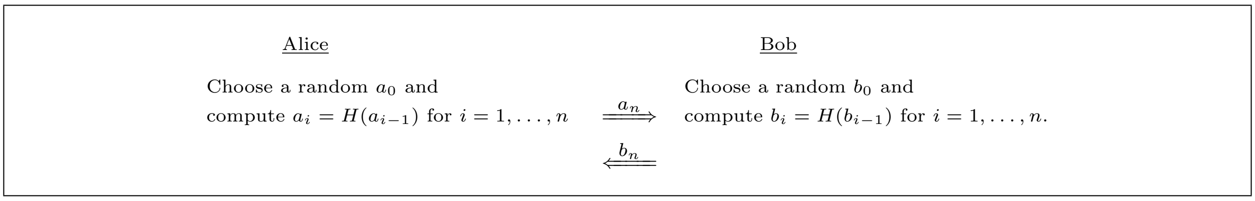
- i, the session counter
- b_{i+1}, the most recently accepted value of Bob's hash chain (hence accept-key(b_{i+1}) has occurred already)
- a one-bit flag, to distinguish the program states **A0** and **A1**.

Similarly, Bob's internal state is:

- i, the session counter
- a_{i+1}, the most recently accepted value of Alice's hash chain (hence accept-key(a_{i+1}) has occurred already)
- a one-bit flag, to distinguish the program states **B0** and **B1**.

Session i of the Jane Doe protocol:

```
A0 (Alice's initial program state) Obtain $m_i$ (possibly from Eve), then
   Commit-message($m_i, i$).
   Compute $d_i = \mathrm{MAC}_{a_i}(m_i)$.
   Send $(d_i, m_i)$; goto A1.
A1 Wait for a message $b'$ (supposedly from Bob), then
   If $H(b') = b_{i+1}$ then
   Let $b_i := b'$, accept-key($b_i$) and send $a_i$. Let $i := i-1$ and goto A0
   else goto A1.
```



Alice

Choose a random a_0 and compute $a_i = H(a_{i-1})$ for $i = 1, \ldots, n$ $\xrightarrow{a_n}$

Bob

Choose a random b_0 and compute $b_i = H(b_{i-1})$ for $i = 1, \ldots, n$.

$\xleftarrow{b_n}$

Figure 4: Initialization Phase of the Jane Doe Recognition Protocol

```
B0 (Bob's initial program state) Wait for a message $(d_i, m_i)$, then send $b_i$ and goto B1.

B1 Wait for a message $a'$ (supposedly from Alice), then
     If $H(a') = a_{i+1}$ then
     Let $a_i := a'$ and accept-key$(a_i)$.
     If $\mathrm{MAC}_{a'}(m_i) = d_i$ then
     Accept $m_i$ as authentic in session $i$
     (else do not accept any message for session $i$).
     Let $i := i - 1$ and goto B0
     else goto B1.
```

Figure 5 depicts the Jane Doe protocol. Lucks et al. first present the Jane Doe protocol in an extended abstract (Lucks et al. 2005), prove its security in a longer version of the paper (Lucks et al. 2008), and finally provided the full version of the paper (Lucks, Zenner, Weimerskirch & Westhoff 2009).

The Jane Doe protocol is proved to be secure given that the preimage resistance, second preimage resistance, and unforgeability properties, and their hash chain equivalents, hold. We note that the following is our formulation of the required properties and not the original formulation presented in Lucks et al. (2005), (2008), and (2009).

Definition 1 *Let secret $y_0, y_1, \ldots, y_i$ and known y_{i+1} be chosen such that $y_{i+1} = H(y_i)$, $y_i = H(y_{i-1}), \ldots,$ $y_1 = H(y_0)$. A hash function H is referred to as a* **depth-i preimage resistant (i-PR)** *hash function when it is infeasible to find y' such that $y_{i+1} = H(y')$.*

Definition 2 *Let secret $y_0, y_1, \ldots, y_{i-1}$ and known y_i, y_{i+1} be chosen such that $y_{i+1} = H(y_i)$, $y_i = H(y_{i-1}), \ldots,$ $y_1 = H(y_0)$. A hash function H is* **depth-i second preimage resistant (i-SPR)** *when it is infeasible to find y', $y' \neq y_i$, such that $y_{i+1} = H(y')$.*

Definition 3 *Let secret $y_0, y_1, \ldots, y_i$ and known y_{i+1} be chosen such that $y_{i+1} = H(y_i)$, $y_i = H(y_{i-1}), \ldots,$ $y_1 = H(y_0)$. A message authentication code MAC is* **depth-i existentially unforgeable** *if it is infeasible to mount an existential forgery against MAC_{y_i} in an adaptive chosen message attack scenario.*

Although the Jane Doe protocol is provably secure, it nonetheless falls short in case of a certain adversarial disruption. In particular, Eve can easily manipulate one party to move forward to the next session, while the other party is still in the previous session. In such a case, a party could get trapped in a state and never be able to finish execution of a session; as a result, he or she remains stuck in that state forever.

3.1.1 Unrecoverability Problem of the Jane Doe Protocol

There is a small time frame associated with each session i. In particular, a message m_i is *fresh* if it is sent within the associated time frame of session i. It is assumed that during each time frame, Alice commits to only one message and Bob accepts at most one message. As a result, the time frame should be known to both Alice and Bob. However, the value i, which could indicate the appropriate time frame, is contained in the internal states of Alice and Bob. Note that i is not being transmitted during the protocol execution and it is implicit that Alice's and Bob's internal states agree on this value. This may be problematic in different ways. First, how will Alice and Bob remain synchronized during the different time frames? Assuming a secure synchronized clock is a quick fix to this problem. However, assuming availability of such a service may not be practical for most ad hoc network scenarios. In particular, Lucks et al. assume that no securely synchronized clock is available. Hence, the process of synchronization is highly dependent on the schedule of received and sent messages, that is, on the dynamics of the communication in the network. This gives rise to the second problem: in case of communication failure or adversarial disruption, this protocol is not equipped with a practical resynchronization process.

We observe that although the Jane Doe protocol is provably secure, it nonetheless falls short in case of the following adversarial disruption. Eve can easily manipulate one party to move forward to the next session, while the other party is still in the previous session. In such a case, a party could get trapped in a state and never be able to finish execution of a session; as a result, he or she remains stuck in that state forever. It is also mentioned in Lucks et al. (2008) that Eve is able to stretch a session at her will.

Figure 6 illustrates a situation where Bob is trapped by Eve in program state **B1**. The condition in program state **B1** fails since $a_{i+1} \neq H(a'_i)$. This will cause Bob to stay in **B1** waiting for a new a_i. Now even if Alice sends him a legitimate message m_i, he will ignore it. Although this looks like a denial of service attack, it is much stronger than that. Eve can go away and yet Alice and Bob are still unable to communicate because Bob is trapped. The details of the disruption are as follows.

Eve sends m'_i and d'_i to Bob and he will automatically decrement his index to i while Alice does not. Eve chooses a'_i such that $a_{i+1} \neq H(a'_i)$, which will make Bob wait for a new a_i. While he is waiting for a new a_i, he will not accept

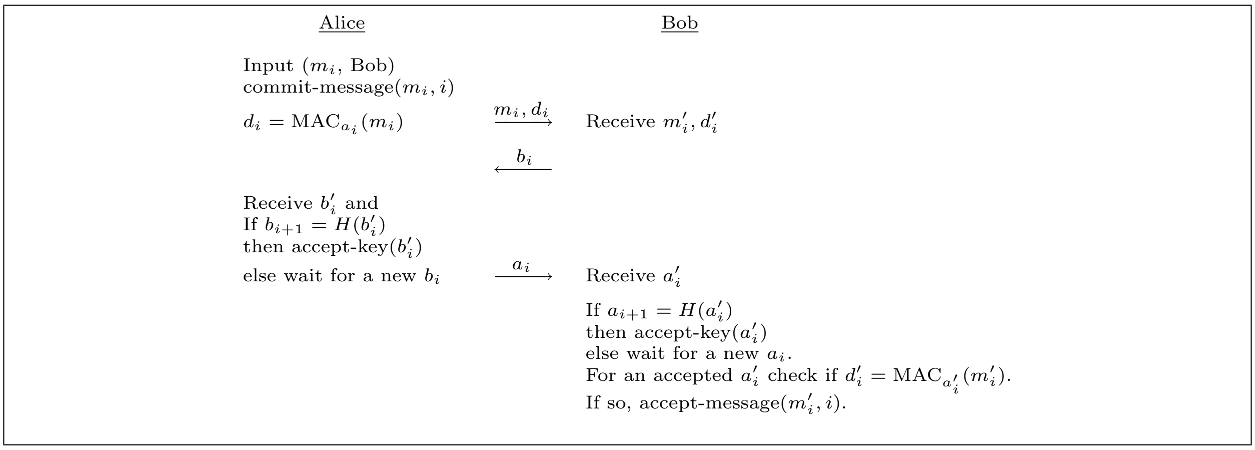


Figure 5: The Jane Doe Entity and Message Recognition Protocol

a message of the form (m_j, d_j), for any j. Hence, even if Alice sends him a legitimate message, he will ignore it. As a result, he is "trapped" in state **B1**.

Lucks et al. (2008) suggest that Bob sends b_i again after he has waited for too long to receive the correct a_i. However, when Alice has not initiated the session and is not anticipating b_i, it is not clear what she is supposed to do. Hence, this will not help the protocol recover in case of this particular disruption.

Eve can play the same trick with Alice and trap her in program state **A1** for an indeterminate period of time; Figure 7 illustrates this situation.

Once again, we note that this inability to recover is a problem since the adversary does not need to continue her active involvement. She can leave the network and yet Alice and Bob will no longer be able to have successful communication. This renders the protocol unusable in practice.

There should be a mechanism to help Alice and Bob resynchronize after having waited for a sufficiently long period of time for a new a_i or b_i. Otherwise, the protocol cannot be resumed and recoverability is lost. One way to perform this resynchronization is to utilize the authenticated channel occasionally. The advantage of this solution is that it is very simple. However, the authenticated channel is expensive and it may not be practical to assume that it is accessible after the initialization phase. For instance, the devices may be widely dispersed, and it may not be possible to collect them again to perform this kind of resynchronization. Furthermore, periodic employment of the resynchronization process, according to a predefined schedule, will not be based on the dynamics of the network. For instance, some devices may be more active than others or there may be more noise present in some parts of the network compared to other parts of the network. Indeed, there is more disruption caused by noise or communication failure in busier parts of the network. Hence, resynchronization among some users may be necessary more often than others. As a result, it is desirable to execute the resynchronization process when it is needed according to the state of the network. We propose the following protocol to overcome these shortcomings. We use the same hash function, H, used by Lucks et al. and write H^j, $j \geq 1$, to denote the case when the hash function H is applied j times iteratively.

4 An Improved MRP with Resynchronization Process

In this section, we present our first improvement on the Jane Doe protocol.

The internal state of Alice and Bob includes:

- i_A (held by Alice) and i_B (held by Bob), counters pointing to the position of Alice and Bob in their respective hash chains,
- $i_{acceptA}$, a counter kept by Alice which is the smallest index such that Alice has accepted the key $b_{i_{acceptA}}$ in session $i_{acceptA}$. Similarly, $i_{acceptB}$, a counter kept by Bob which is the smallest index such that Bob has accepted the key $a_{i_{acceptB}}$ in session $i_{acceptB}$.

Alice executes the protocol as follows:

- Let $i := i_A$ and $j_A := i_{acceptA} - i_A$;
- Wait for m_i (possibly from Eve), then
- commit-message(m_i, i);
- compute $d_i = \text{MAC}_{a_i}(i \| m_i)$;
- send $(i \| m_i, d_i)$;
- wait for a message b'_i (supposedly from Bob), then if $H^{j_A}(b'_i) = b_{i_{acceptA}}$, (key verification step) then $b_i := b'_i$; accept-key(b_i); send a_i; set $i_{acceptA} := i$ and $i_A = i - 1$;
  else initiate the resynchronization process.

Bob executes the protocol as follows:

- Let $j_B := i_{acceptB} - i_B$;
- Wait for a message $(i' \| m'_{i'}, d_{i'})$.
- If $i' = i_B$, then send $b_{i'}$, else initiate the resynchronization process.

Eve		Bob
Choose random m'_i and d'_i.	$\xrightarrow{m'_i, d'_i}$	Move to the next time-frame upon reception of the new message.
	$\xleftarrow{b_i}$	
Choose a'_i such that $a_{i+1} \neq H(a'_i)$.	$\xrightarrow{a'_i}$	Since $a_{i+1} \neq H(a'_i)$, wait for a new a_i.

Figure 6: Eve “trapping” Bob in state **B1**

Alice		Eve
Input (m_i, Bob). commit-message(m_i, i).		
Compute $d_i = \text{MAC}_{a_i}(m_i)$.	$\xrightarrow{m_i, d_i}$	
Since $b_{i+1} \neq H(b'_i)$, wait for a new b_i.	$\xleftarrow{b'_i}$	Choose b'_i such that $b_{i+1} \neq H(b'_i)$.

Figure 7: Eve “trapping” Alice in state **A1**

- Wait for a message $a'_{i'}$ (supposedly from Alice), then
  if $H^{j_B}(a'_{i'}) = a_{i_{acceptB}}$, (key verification step)
  then $a_{i'} := a'_{i'}$; accept-key$(a_{i'})$; set $i_{acceptB} := i'$ and $i_B := i' - 1$
  if $\text{MAC}_{a_{i'}}(i' \| m'_{i'}) = d_{i'}$
  then accept $m'_{i'}$ as authentic in session i';

  else initiate the resynchronization process.

Figure 8 illustrates this protocol. Let us first highlight the differences between this protocol and the Jane Doe protocol of Lucks et al. In the internal states of Alice and Bob, the session counter i is replaced by i_A and i_B to incorporate the adversarial ability to manipulate a party to decrement the session counter, as was discussed previously, and consequently change its position in the hash chain. For the same reason, $i+1$ is changed to $i_{acceptA}$ and $i_{acceptB}$ as the smallest index such that a key has been accepted by Alice or Bob, respectively. Moreover, a_{i+1} and b_{i+1} are replaced by $a_{i_{acceptB}}$ and $b_{i_{acceptA}}$ as the accepted keys. Further, parameters j_A and j_B are introduced to deal with the case where $i_{acceptA} > i_A{+}1$ or $i_{acceptB} > i_B{+}1$, respectively, due to an adversary's intrusions. A related modification refers to the *key verification* step, where the users may need to apply the hash function H more than once. In the Jane Doe protocol, the session counter is not being transmitted or committed to by either party. However, we require that Alice commits to i_A and transmits it in the first flow. This allows Bob to easily detect any possible manipulations of the session counter by Eve. Furthermore, we provide a resynchronization process, allowing Alice and Bob to initiate the resynchronization process when they do not receive the correct keys. Hence, the adversary can no longer “trap” them in states **A1** or **B1**, as was explained previously.

Surely, it holds that $i_A = i_B$ when the adversary has been passive since the initialization. Moreover, in the case where all flows are safely relayed from the initialization, Alice and Bob will accept every single key from the other party and move forward in the hash chain together. Hence, in the ith session, $i_A = i_B = i$ and $i_{acceptA} = i_{acceptB} = i + 1$. In particular, $j_A = i_{acceptA} - i_A = 1$ and $j_B = i_{acceptB} - i_B = 1$. However, once the adversary begins sending messages to Alice and Bob, she is capable of manipulating either party to decrement their session counter in a bogus session. Hence, Alice and Bob will need to resynchronize to agree on a mutual position in their respective hash chains, which may result in $j_A \neq 1$ or $j_B \neq 1$.

Note that this variant is instructing Alice and Bob to initiate resynchronization whenever a mismatch occurs. Hence, once the adversary initiates a bogus session, she can no longer continue another bogus session undetected. That is, she can make Alice and Bob decrement their session counters at most once. Hence, we have $|i_A - i_B| \leq 1$. In case of an active intrusion, the participants are not supposed to accept the key Eve sends them and, as a result, the values of $i_{acceptA}$ and $i_{acceptB}$ are not going to be updated. Consequently, we obtain $j_A = i_{acceptA} - i_A = 2$ or $j_B = i_{acceptB} - i_B = 2$.

In this protocol, the session counter is being transmitted in the first flow. Moreover, Alice commits to this value as part of the message, so the adversary cannot arbitrarily change it without being detected. This implies that the security proof of the Lucks et al. protocol will apply to this new variant as well. Furthermore, once either user realizes that Eve could have manipulated the values, they can initiate a resynchronization process. This process allows them to agree on a session counter $i_A = i_B$, which indicates the corresponding position of each user in their respective hash chains.

4.1 Resynchronization Process

At some point during the execution of the protocol, either Alice or Bob realizes the need for resynchronizing with the other party. This may be due to a mismatch caused by adversarial efforts or just due to some communication failure or noise. In the resynchronization process, Alice

Alice		Bob

Internal-state= i_A and $i_{acceptA}$
Let $i := i_A$ and $j_A := i_{acceptA} - i_A$
Receive input (m_i, Bob) and
commit-message(m_i, i)
		Internal-state= i_B and $i_{acceptB}$
		Let $j_B := i_{acceptB} - i_B$;

$d_i = \mathrm{MAC}_{a_i}(i \| m_i)$	$\xrightarrow{i, m_i, d_i}$	Receive $i', m'_{i'}, d'_{i'}$
		If $i' = i_B$, then send $b_{i'}$,
	$\xleftarrow{b_{i'}}$	else initiate resynchronization.
Receive b'_i and If $b_{i_{acceptA}} = H^{j_A}(b'_i)$, then accept-key($b'_i$),		
send a_i, and	$\xrightarrow{a_i}$	Receive $a'_{i'}$
let $i_{acceptA} := i$ and $i_A = i - 1$; else initiate resynchronization		If $H^{j_B}(a'_{i'}) = a_{i_{acceptB}}$, then accept-key($a'_{i'}$) and, let $i_{acceptB} := i'$ and $i_B := i' - 1$ else initiate resynchronization. For an accepted $a'_{i'}$ check if $d'_{i'} = \mathrm{MAC}_{a'_{i'}}(i' \| m'_{i'})$. If so, accept-message($m'_{i'}, i'$).

Figure 8: Our Proposed Variant of the Jane Doe Protocol

computes

$$I_A := \min\{i : \text{ Alice has revealed } a_i\} - 1$$

and, similarly,

$$I_B := \min\{i : \text{ Bob has revealed } b_i\} - 1$$

is computed by Bob. Then, they exchange I_A and I_B over the insecure channel. Note that, Eve can change these values, say to I'_A and I'_B, since they are being sent over the insecure channel.

Recall that we are instructing Alice and Bob to resynchronize whenever they notice a mismatch. This implies that the adversary, or some noise in the channel, can make them increment their session counters at most once before they try to resynchronize again. Hence, we have $|I_A - I_B| \leq 1$. This fact alone does not enable Alice and Bob to detect Eve's manipulation with I_A and I_B. However, it makes it impossible for Eve to choose values for I'_A and I'_B which are smaller than $I_A - 1$ and $I_B - 1$, respectively. We emphasize that this feature is important here. In the absence of such a feature, Eve can choose I'_A and I'_B to be very small and exhaust the hash chains too quickly. That would constitute a strong denial of service attack that can be prevented as follows.

Alice checks to make sure $|I_A - I'_B| \leq 1$ and Bob checks to see if $|I'_A - I_B| \leq 1$ holds. If either of these do not hold, it means that the adversary is attempting to intrude while they are trying to resynchronize. If $|I_A - I'_B| \leq 1$ and $|I'_A - I_B| \leq 1$ hold, then Alice and Bob will let $i_A := \min(I_A, I'_B)$ and $i_B := \min(I'_A, I_B)$, respectively. Figure 9 depicts the resynchronization process.

Note that an active adversary can always disrupt the synchronization. When one party realizes this, he or she will call for a resynchronization again. If the adversary is passive in the resynchronization stage, then $I_A = I'_A$ and $I_B = I'_B$. As a result, $i_A = i_B$ and synchronization is achieved.

However, we will show that intrusions of an active adversary during the resynchronization stage, resulting in $i_A \neq i_B$, are going to be detected by either Alice or Bob. In case of intrusions where $|I_A - I'_B| > 1$ or $|I'_A - I_B| > 1$, the adversary is detected right away as discussed above. The rest of the intrusions are detected in the first session of the protocol immediately after the resynchronization, depicted in Fig 10. We show that the adversary is detected unless she has found unrevealed preimages of particular values in the hash chain. Note that i_A and i_B cannot differ very much, due to the conditions $|I_A - I'_B| \leq 1$ and $|I'_A - I_B| \leq 1$. However, we can prove the same statement even if the difference between i_A and i_B is not bounded.

In order for Eve not to be detected by Bob in the key verification step, she must replace a_{i_A} with a_{i_B}. Otherwise, Bob will not accept the key and he will initiate resynchronization regardless of the values of m_{i_B} and d_{i_B}. Similarly, she has to replace b_{i_B} with b_{i_A}, otherwise, she will be detected by Alice. Now, assume that $i_A < i_B$ after the resynchronization. Finding a correct value for b_{i_A} means that Eve has found a nonempty chain of preimages $(b_{i_B}, b_{i_B-1}, \ldots, b_{i_A+1})$. Similarly, if $i_A > i_B$ and the adversary goes undetected, she has found a chain of preimages $(a_{i_A}, a_{i_A-1}, \ldots, a_{i_B+1})$. Hence, as long as finding preimages of H is a hard task, the adversary will be detected with high probability. As a result, we obtain the following theorem.

Theorem 2 *Let H be a depth-i second preimage resistant and depth-i preimage resistant hash function in the protocol of Figure 8. Consider a polynomially bounded adversary who changes the values of I_A or I_B in the resynchronization process of Figure 9, resulting in $i_A \neq i_B$. An undetected such intrusion can only occur with a negligible probability.*

As a small efficiency improvement, we note that it is not necessary for Alice to store both i_A and $i_{acceptA}$. It would suffice to store i_A and j_A, because $i_{acceptA} = i_A + j_A$. Since

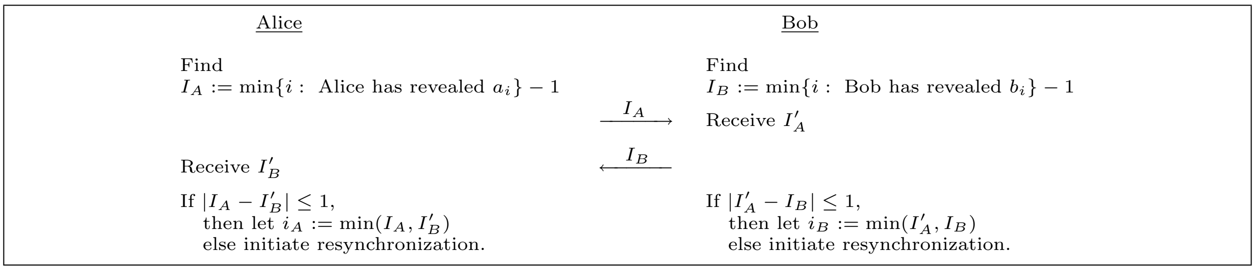

Alice
Bob
Find
$I_A := \min\{i :$ Alice has revealed $a_i\} - 1$
Find
$I_B := \min\{i :$ Bob has revealed $b_i\} - 1$
$\xrightarrow{I_A}$
Receive I'_A
$\xleftarrow{I_B}$
Receive I'_B
If $|I_A - I'_B| \le 1$,
then let $i_A := \min(I_A, I'_B)$
else initiate resynchronization.
If $|I'_A - I_B| \le 1$,
then let $i_B := \min(I'_A, I_B)$
else initiate resynchronization.

Figure 9: Resynchronization Process for the Proposed Protocol

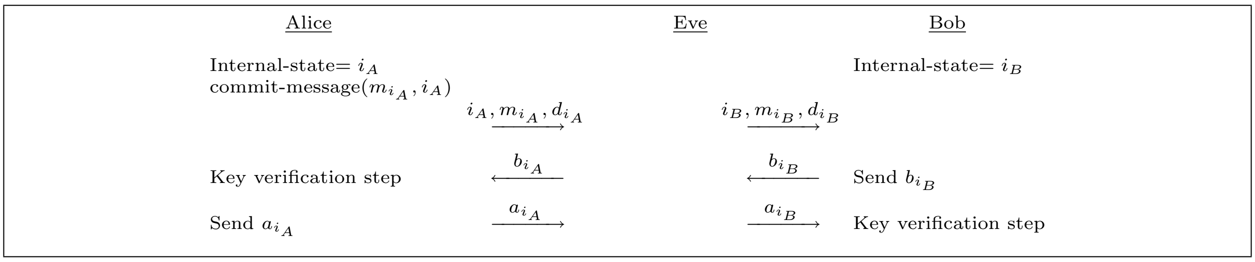


Figure 10: The First Execution after the Resynchronization

$j_A = 1$ or 2, this reduces the amount of information that Alice needs to store. A similar remark holds for Bob.

Although the improved version of the protocol along with the resynchronization protocol fully recovers from these noted disruptions, it is of interest to design a protocol that recovers on its own, without having to call upon an external protocol. That is, a protocol that attains self-recoverability is usually preferred to a protocol that depends on a second protocol when trying to recover.

In the next section, we propose a message recognition protocol which attains self-recoverability in case of the noted disruptions. It is in fact a highly nontrivial task to modify the protocol to achieve self-recoverability. Because the entities may be in additional "states", depending on the information they possess and its authenticity, the protocol is necessarily more complicated. As a consequence, the security proof is more difficult.

5 A New MRP with Self-recoverability

We describe the details of our proposed MRP in this section, while the security and recoverability analyses are postponed to the next session. Although this protocol is based on the Jane Doe protocol proposed by Lucks et al., the logic of the instructions of Alice and Bob has changed considerably. Moreover, the information exchanged between Alice and Bob has changed as well.

Note that each pair of users can execute this new protocol. However, as in the Jane Doe protocol, there must be a different pair of hash chains for each pair of communicating users. It is implicitly assumed that Alice and Bob are the communicating parties in the rest of the paper.

The initialization phase and the setup of the hash chains are exactly as in the Jane Doe protocol. The internal state of Alice includes (along with each variable's initial value):

- $i_A := n - 1$: the position of Alice in her chain.
- $i_{acceptA} := n$: the last index of Bob's chain that was accepted by Alice.
- $b_A := b_n$: the last value of Bob's chain that was accepted by Alice.
- $M := Null$: the input message to be authenticated in the current session.
- a one-bit flag, to distinguish the program states **A0** and **A1**.

Similarly, Bob's internal state is as follows:

- $i_B := n - 1$: the position of Bob in his chain.
- $i_{acceptB} := n$: the last index of Alice's chain that was accepted by Bob.
- $a_B := a_n$: the last value of Alice's chain that was accepted by Bob.
- $e' := Null$: the MAC value supposedly received from Alice.
- $M' := Null$: the message supposedly received from Alice.
- a one-trit flag, to distinguish the program states **B0**, **B1**, and **B2**.

Alice and Bob start in program states **A0** and **B0**.

We write commit-message(M, i_A) to indicate that Alice is committing herself to sending the message M to Bob in session i_A. We let T be the maximum amount of time Alice waits to receive a response from Bob, and vice versa.

A0 is executed as follows:

```
If $i_A \le 0$ then **Abort**.

Receive input $(M)$ and commit-message$(M, i_A)$.

Compute $e_{i_A} := \mathrm{MAC}_{a_{i_A}}(i_A \| M)$.

Send $[e_{i_A}, M]$ to Bob and **goto A1**.
```

```
B0 is executed as follows:
  If $i_B \leq 0$ then Abort.
  Wait to receive $[e', M']$, then goto B1.
B1 has the following description:
  Send $[i_B, b_{i_B}]$ to Alice and goto B2.
A1 is performed in the following manner:
  Wait at most time $T$ to receive $[i'_B, b']$.
  If $[i'_B, b']$ is received, then
    If $i'_B = i_{acceptA}$ and $b_A = b'$ (Bob has not received the last flow of the previous session) then
      Let $N := Null$.
      Send $[i_{acceptA}, a_{i_{acceptA}}, N]$ and goto A0.
    If $i'_B = i_A$ and $b_A = H(b')$ then (Alice and Bob seem to be synchronized.)
      Let $N := M$.
      Send $[i_A, a_{i_A}, N]$ to Bob.
      Let $i_{acceptA} := i'_B$, $b_A := b'$ and $i_A := i_A - 1$.
      (Alice updates her state.)
      goto A0.
    else Resend $[e_{i_A}, M]$ to Bob and goto A1.
  If timeout then
    Resend $[e_{i_A}, M]$ to Bob and goto A1.
B2 is performed as follows:
  Wait at most time $T$ to receive $[i'_A, a', N']$.
  If $[i'_A, a', N']$ is received, then
    If $i'_A = i_B$ and $a_B = H(a')$ then (Alice and Bob seem to be synchronized.)
      If $N' = M'$ and $e' = \mathrm{MAC}_{a'}(i'_A \| M')$ then
        Accept($M'$, $i_B$).
      else Accept($Null$).
      Let $i_{acceptB} := i'_A$, $a_B := a'$ and $i_B := i_B - 1$.
      (Bob updates his state.)
      goto B0.
    else goto B1.
  If timeout, then goto B1.
```

Figure 11 illustrates the main steps of this protocol. For simplicity, the instructions on what to do in case one party does not receive any response from the other party is not included in the figure.

If either Alice or Bob receives a message that they did not expect, they are going to ignore it. For instance, while Alice is in state **A1** and is waiting to receive a message of the form (i_B, b), she is going to ignore messages of the form (M') that request for a new session and correspond to state **A0**. Analogously, when Bob is in state **B2**, he is waiting for a message of type i_A, a, N. He is going to ignore messages of the form e'_{i_A}, M' since they correspond to state **B0**. In general, each party only acts on received messages that have the expected structure in accordance to their current program state.

When Alice is waiting in state **A1** for Bob to respond, she is set to wait for time T. If she receives a message i'_B, b' in time T, then she processed it in state **A1**, and otherwise, she resends e_{i_A}, M to Bob. Similarly, Bob waits to receive a message i'_A, a', N', supposedly from Alice, for time T. If he does not receive such a message, he resends i_B, b to Alice.

Note that Eve can block the last flow of Alice, i_A, a, N. In this case, Alice has decremented her state, while Bob is waiting to receive i_A, a, N, and possibly resending i_B, b_{i_B} to remind Alice to send him i_A, a, N. However, since Alice has moved her state to **A0**, she will ignore Bob's messages. This may appear to be problematic since Bob is waiting for Alice. However, once Alice is ready to authenticate a new message to Bob, she will be in program state **A1** again, and communication will resume.

6 Security of Our New MRP with Self-recoverability

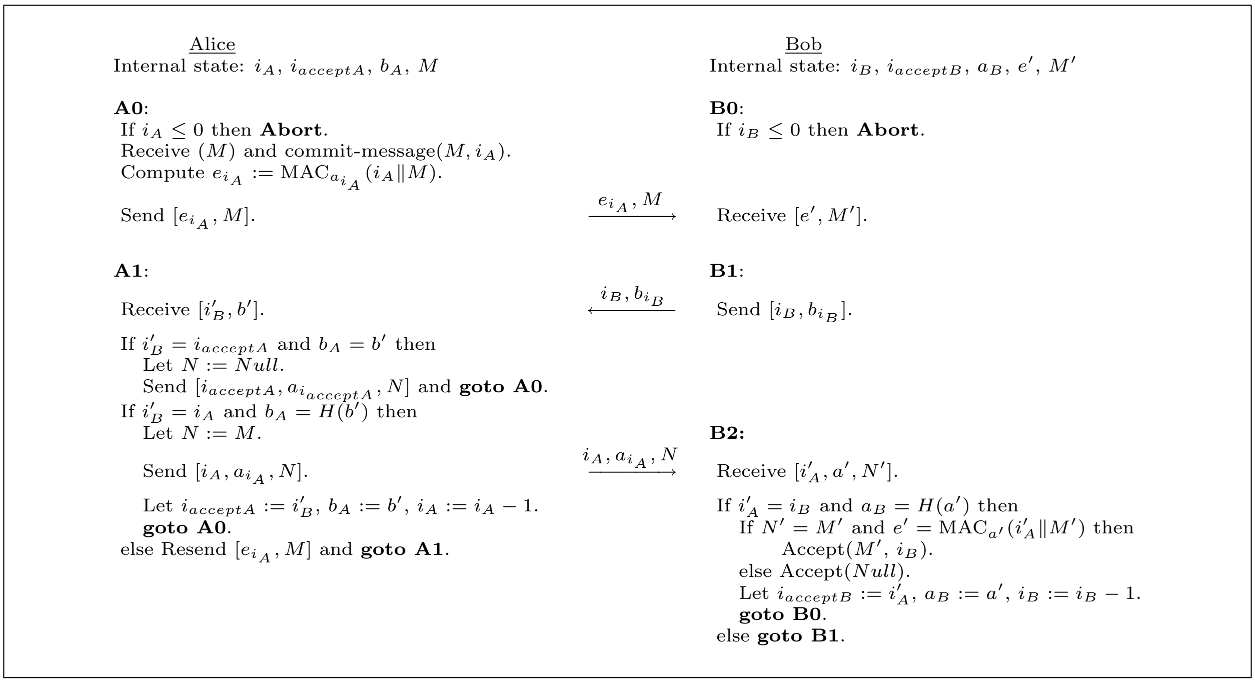
In this section, we consider different types of possible attacks against our protocol. Finally, we conclude with a theorem which ensures the security of our protocol.

6.1 Single-session Attacks

In this section, we consider attacks that are started and completed in a single session. We assume that Eve has stayed passive all along and she becomes active in the current session for the first time. In case of a successful attack, Bob will accept some message M' in the same session, where M' is not $Null$ and not the message sent by Alice in that session. Since Eve has been passive before this session, we will have $i_A = i_B$ at the start of the session; we let $i := i_A = i_B$ for ease of reference. For the same reason, we have $i_{acceptA} = i_{acceptB} = i + 1$. Furthermore, Alice and Bob will have accepted all the intended keys so far. That is, $a_B = a_{i+1}$ and $b_A = b_{i+1}$.

We now want to exhaustively list all possible single-session attacks. We follow the notation of Gehrmann (1998) in referring to different orderings of the flows. In each attack, the adversary sends a flow to either Alice or Bob and receives a flow in response. This notation labels a flow by **A** if the recipient is Alice, or by **B** when the recipient is Bob. For instance, the following attack scenario corresponds to the attack type of ABAB:

- **A**: Eve sends M to Alice and she responds with e_{i_A}, M.
- **B**: Eve sends e', M' to Bob and he replies with i_B, b_{i_B}.



```
Alice                                                       Bob
Internal state: $i_A$, $i_{acceptA}$, $b_A$, $M$            Internal state: $i_B$, $i_{acceptB}$, $a_B$, $e'$, $M'$

A0:                                                         B0:
 If $i_A \leq 0$ then Abort.                                 If $i_B \leq 0$ then Abort.
 Receive ($M$) and commit-message($M, i_A$).
 Compute $e_{i_A} := \mathrm{MAC}_{a_{i_A}}(i_A \| M)$.
 Send $[e_{i_A}, M]$.              $\xrightarrow{e_{i_A}, M}$           Receive $[e', M']$.

A1:                                                         B1:
 Receive $[i'_B, b']$.             $\xleftarrow{i_B, b_{i_B}}$          Send $[i_B, b_{i_B}]$.
 If $i'_B = i_{acceptA}$ and $b_A = b'$ then
    Let $N := Null$.
    Send $[i_{acceptA}, a_{i_{acceptA}}, N]$ and goto A0.
 If $i'_B = i_A$ and $b_A = H(b')$ then
    Let $N := M$.                                           B2:
    Send $[i_A, a_{i_A}, N]$.      $\xrightarrow{i_A, a_{i_A}, N}$      Receive $[i'_A, a', N']$.
    Let $i_{acceptA} := i'_B$, $b_A := b'$, $i_A := i_A - 1$.   If $i'_A = i_B$ and $a_B = H(a')$ then
    goto A0.                                                   If $N' = M'$ and $e' = \mathrm{MAC}_{a'}(i'_A \| M')$ then
 else Resend $[e_{i_A}, M]$ and goto A1.                            Accept($M'$, $i_B$).
                                                               else Accept($Null$).
                                                               Let $i_{acceptB} := i'_A$, $a_B := a'$, $i_B := i_B - 1$.
                                                               goto B0.
                                                             else goto B1.
```

Figure 11: Our Proposed Message Recognition Protocol (Common Case)

- **A**: Eve sends i'_B, b' to Alice and receives i_A, a_{i_A}, N from her.
- **B**: Eve sends i'_A, a', N' to Bob.

The number of distinct attacks against a three flow protocol is proved to be $\binom{4}{2} = 6$ in Gehrmann (1998). These attacks are denoted AABB, ABBA, BABA, ABAB, BBAA, and BAAB. We will look at these different attacks separately. We stress that Gehrmann (1998) formally proves this list to be an exhaustive list of all possible types of attacks.

One can show that the BABA attack scenario can be reduced to the ABBA attack. That is, if an adversary Oscar can mount a successful attack of type BABA, then Eve can use Oscar and succeed in the ABBA attack scenario. Similarly, we can show that the BAAB and ABBA attack scenarios are reduced to the ABAB case. It remains to analyze the other three attack scenarios, namely AABB, BBAA, and ABAB. We will reduce a successful adversary in these attacks to a player who can mount a depth-i existential forgery or can find depth-i preimages or depth-i second preimages.

6.1.1 Attack of Type AABB

Figure 12 depicts an attack of type AABB.

If $i'_A \neq i_B$, Bob will not accept any messages. Since $i_A = i_B = i$, Eve has to set $i'_A := i_A$ in order to succeed. Moreover, Alice reveals i_A and a_{i_A} only if b' is verified; that is, if $b_A = H(b')$ (note that $b_A = b_{i+1}$, as discussed before).

Eve first interacts with Alice and has to find b' before seeing $b_{i_B} = b_i$. This implies that she has found a preimage of $b_A = b_{i+1}$. This exactly translates to the notion of i-PR defined in Def. 1.

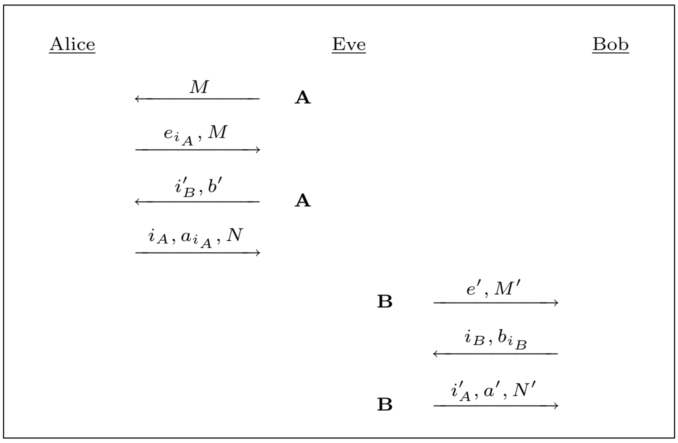


Figure 12: Attack of Type AABB

6.1.2 Attack of Type BBAA

Figure 13 illustrates the attack of type BBAA.

Alice tries to deceive Bob before she starts interacting with Alice. In order to succeed, Eve needs to present Bob with an a' such that $a_B = H(a')$, without having seen $a_{i_A} = a_i$ (note that $a_B = a_{i+1}$, as discussed before). In other words, she is trying to find a preimage of $a_B = a_{i+1}$. If Eve can successfully find such a preimage, the she translates to a successful player who finds depth-i preimages, as defined in Def. 1.

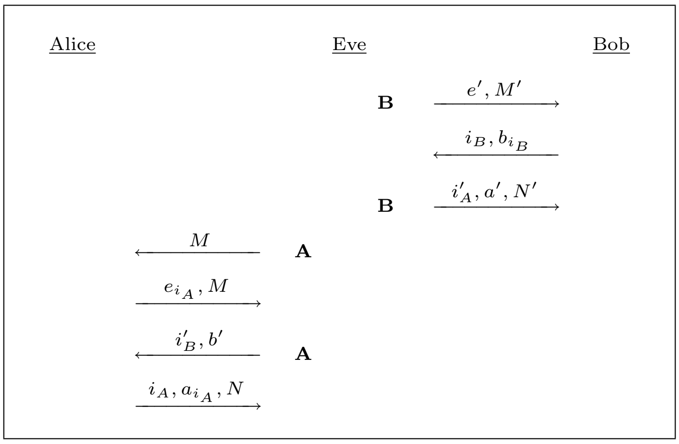


Figure 13: Attack of Type BBAA

6.1.3 Attack of Type ABAB

Depicted in Fig. 14 is the ABAB attack.

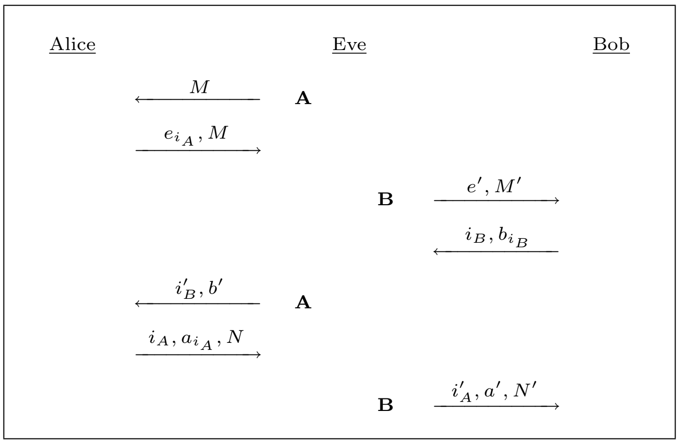


Figure 14: Attack of Type ABAB

In this scenario, Eve receives $b_{i_B} = b_i$ before she has to send b' to Alice. We analyze the two cases $b' = b_i$ and $b' \neq b_i$ separately.

If $b' \neq b_i$, then it implies that Eve has found a depth-i second preimage of $b_A = b_{i+1}$.

Otherwise, $b' = b_i$. Alice will verify $b' = b_i$ and reveal $a_{i_A} = a_i$. Eve now has two choices. She chooses a' such that either $a' = a_{i_A}$ or $a' \neq a_{i_A}$. If $a' \neq a_{i_A}$, then she has found a depth-i second preimage of $a_{i+1} = a_B$. On other hand, if $a' = a_{i_A}$, then for Eve to succeed, she must set $N' := M'$ and she must have set $e' := \mathrm{MAC}_{a'}(i'_A \| M')$ *before* learning a'. That is, Eve has successfully forged a MAC. This reduces to the notion of depth-i existential forgery defined in Def. 3.

6.1.4 Reducing the BABA attack to an ABBA attack

The ABBA attack scenario, depicted in Fig. 15, is as follows:

- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.
- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **B**: Oscar sends i'_A, a', N' to Bob.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.

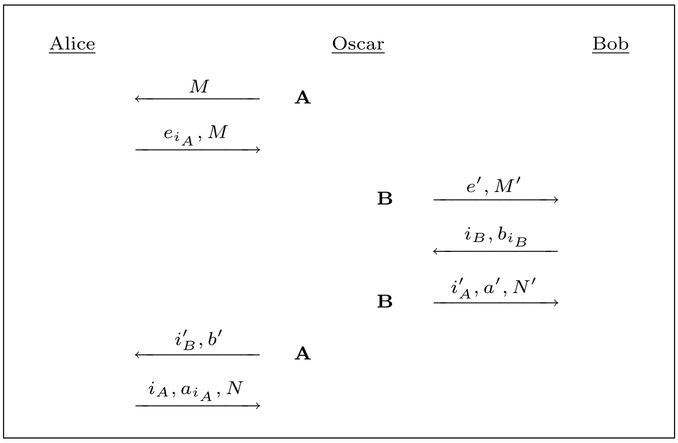


Figure 15: Attack of Type ABBA

On the other hand, the BABA attack scenario, illustrated in Fig. 16, is as follows:

- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.
- **B**: Oscar sends i'_A, a', N' to Bob.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.

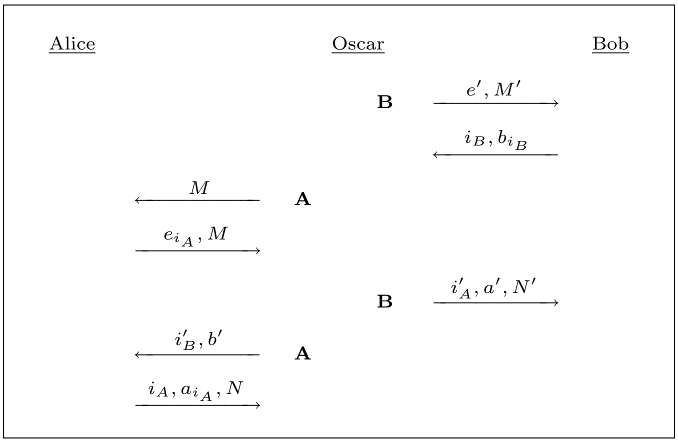


Figure 16: Attack of Type BABA

These two attack scenarios differ in the order of the first two steps and are identical otherwise. In the BABA attack scenario, Oscar commits to e' and M' before receiving e_{i_A}. Note that knowing e_{i_A} could possibly help him in choosing e'. On the other hand, Oscar receives i_B and b_{i_B} before sending M. The adversary knows the value of i_B. Moreover, the choice of M is independent of the value of b_{i_B}. In other words, knowing b_{i_B} is not going to help the adversary in choosing M. Hence, if Oscar can win in the BABA attack scenario by first committing to e' and M' and then receiving e_{i_A}, then he can win the ABBA attack scenario with the same values M, M', and e.

6.1.5 Reducing the ABBA attack to an ABAB attack

Recall the ABAB attack scenario:

- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.
- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.
- **B**: Oscar sends i'_A, a', N' to Bob.

The ABBA attack differs from the ABAB attack in the order of the last two steps. In the ABAB attack, Oscar receives i_A, a_{i_A}, N from Alice, and then he has to send i'_A, a', N' to Bob. Knowing i_A, a_{i_A}, N can help him choose a winning i'_A, a', N', whereas in the ABBA attack scenario, Oscar sends i'_A, a', N' before seeing i_A, a_{i_A}, N. If Oscar has a winning strategy in the ABBA attack scenario, then using the same values of i'_A, a', N', he will win the ABAB attack scenario.

6.1.6 Reducing the BAAB attack to an ABAB attack

The BAAB attack scenario is as follows:

- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.
- **B**: Oscar sends i'_A, a', N' to Bob.

Figure 17 depicts this attack.

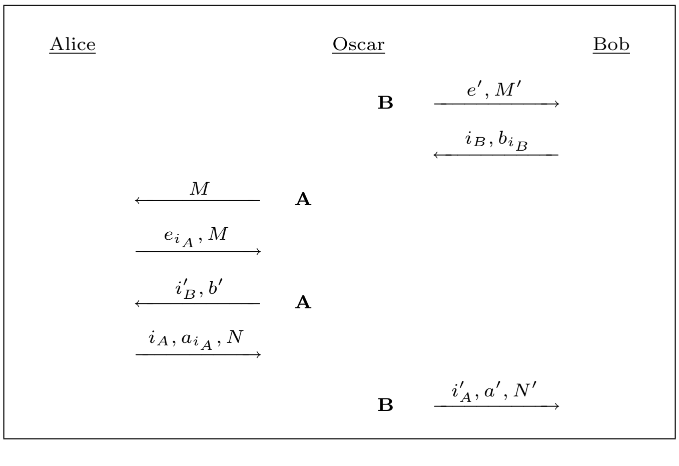


Figure 17: Attack of Type BAAB

The analysis of this case is analogous to that of Section 6.1.4. The BAAB attack scenario differs from the ABAB attack scenario in the order of the first two steps. In the BAAB attack scenario, Oscar has to commit to e' and M' before seeing e_{i_A}. Although Oscar receives i_B and b_{i_B} before sending M, these values are independent of the choice of M. That is, seeing b_{i_B} is not going to help the adversary in choosing M. Hence, a winning strategy in the BAAB attack scenario reduces to a winning strategy in the ABAB attack scenario.

6.2 Multi-session Attacks

Having ruled out the possibility of single-session attacks, we now turn our attention to multi-session attacks. Consider attack scenarios which occur over two or more sessions. In such a case, the adversary becomes active in one session and concludes her attack in one of the following sessions. In case of a successful attack, Bob will accept M' in the last session of the attack, where M' is not $Null$ and not the message sent by Alice in that session.

Just before Eve becomes active, similar to the single-session attack scenario discussed above, we must have $i_A = i_B$ and $i_{acceptA} = i_{acceptB} = i_A + 1$. We again let $i := i_A = i_B$ for ease of reference. Moreover, all of the intended keys will have been accepted to this point, so as a result, $a_B = a_{i+1}$ and $b_A = b_{i+1}$.

We now assume that during session i, Eve becomes active by initiating a flow with either Alice or Bob, or changing the information sent by them. Since we are considering multi-session attacks, the attack should not entirely take place in one session. As a result, Eve is not making Bob accept her message M' immediately after she becomes active. The following three cases can happen once Eve becomes active:

Case 1. Bob is not engaged right away. That is, Eve first interacts with Alice.

Case 2. Bob is engaged right away and he outputs the message M, sent by Alice.

Case 3. Bob is engaged right away and he outputs $Null$.

We discuss each case separately.

Case 1. Let us assume that Eve first interacts with Alice and does not engage Bob. In order for Alice to conclude her session, she must receive i'_B, b' such that $i'_B = i$ and $b_{i+1} = H(b')$. Otherwise, Alice will detect that something is going on, hence, she will not reveal i, a_i and, instead, will resend e_i, M. If Eve wants to remain undetected and be able to continue with her attack, she needs to send i'_B, b' such that $i'_B = i$ and $b_{i+1} = H(b')$. This means that Eve has found a depth-i preimage of b_{i+1}.

Case 2. Now assume that Bob is engaged and he outputs the message M, sent by Alice. That is, on input (M), Alice has sent e_i, M to Bob. Since Bob accepts M at the end, it means that he, indeed, has received M in the first flow. Moreover, for Bob to accept M, he must receive i'_A, a', N' such that $i'_A = i$, $a_{i+1} = H(a')$, and $N' = M$. There are three different cases to consider here.

- Not having received i, a_i, M from Alice, Eve finds i'_A, a', N' such that $i'_A = i$ and $a_{i+1} = H(a')$. That is, she finds a depth-i preimage of a_{i+1}.
- Having received i, a_i, M from Alice, Eve finds i'_A, a', N' such that $i'_A = i$, $a_{i+1} = H(a')$, and $a_i \neq a'$. That is, she finds a depth-i second preimage of a_{i+1}.
- Eve sets $i'_A, a', N' = i, a_i, M$. That is, Eve relays Alice's last flow. Note that Alice reveals her last flow only if she receives i'_B, b' such that $i'_B = i$ and $b_{i+1} = H(b')$. There are again three cases to consider here. Either Eve has found a depth-i preimage of b_{i+1}, she has found a depth-i second preimage of b_{i+1}, or she has relayed i, b faithfully. In the latter case, Eve has faithfully relayed all messages, and this does not constitute an attack by an active adversary. This contradicts our assumption that Eve first becomes active in session i.

Case 3. Bob is engaged right away and he outputs $Null$. This means that he has received and verified i'_A and a'. There are again three cases to consider. Either Eve has found a depth-i preimage of a_{i+1}, or she has found a depth-i second preimage of a_{i+1}, or i'_A and a' are the correct i, a_i as revealed by Alice. In this last case, Alice and Bob have successfully remained synchronized, but were unable to authenticate the messages they intended to authenticate.

The above discussion concludes that in the session immediately after Eve becomes active, she can only stop Alice and Bob from authenticating the intended message, but she cannot bring them out of their synchronized states unless she is able to solve the depth-i PR or depth-i SPR problems defined in Definitions 1 and 2. Moreover, if Alice and Bob are synchronized at the beginning of a session, then they will end the session in a synchronized state, unless Eve is able to find depth-i preimages or depth-i second preimages.

At the beginning of a multi-session attack, Alice and Bob are synchronized. The above discussion implies that they remain synchronized until the very last session of the attack. We can look at this last session of the attack separately and think of it as a single-session attack. As a result, any multi-session attack translates to a single-session attack, which were already ruled out in Section 6.1.

Note that the adversary can only exhaust Alice's and Bob's values of the hash chain one at a time. That is, she can not make them jump more than one step down the hash chain values.

6.3 Self-recoverability

In this section, we show that once Eve stops interfering with their message flows, Alice and Bob will be able to resume successful communication of recognized messages. Because we have already shown that Alice and Bob remain synchronized in their i values throughout an active attack by Eve (under the security assumptions on H and MAC), we need only show that they do not get "trapped" in a program state, as was the case in the Jane Doe protocol, for example.

We consider the possible combinations of program states which Alice and Bob are in when Eve becomes passive. We first consider the case where Alice is in state **A1**.

- If Alice is in **A1** and Bob is in **B0**, then after time T, Alice will resend $[e_{i_A}, M]$ to Bob, which will cause him to leave state **B0**, and the protocol will continue.
- If Alice is in **A1** and Bob is in **B1**, then Bob will send $[i_B, b_{i_B}]$ to Alice and advance to **B2**, which will cause her to send an appropriate message to Bob, and herself return to **A0**. Bob will return to **B0**, though he may Accept($Null$) if Eve forged the M' which caused Bob to enter the **B1** state. This can of course only affect the first Accept after Eve's interference, however.
- If Alice is in **A1** and Bob is in **B2**, then Alice will be resending useless messages to Bob, and staying in **A1**, but after time T, Bob will return to **B1**, and we proceed as above.

If Alice is in **A0**, then no progress will be made until the next time she tries to send a message to Bob. At that point, Alice will enter state **A1**, and the analysis continues as above.

6.4 Security and Self-recoverability Theorem

The above discussion concludes the discussion of the security and self-recoverability of the proposed message recognition protocol, and forms the proof of the following theorem.

Theorem 3 *A successful adversary against the protocol of Section 5 who efficiently deceives Bob into accepting (M',i), where M' is not $Null$ and Alice did not send M' in session i, implies an efficient algorithm that finds depth-i preimages or depth-i second preimages, or creates depth-i existential forgeries. Moreover, the adversary cannot stop Alice and Bob from successfully executing the protocol unless she is actively disrupting the communication for the lifetime of Alice and Bob.*

7 Explicit Confirmation

In this section, we propose a new protocol that provides explicit confirmation. To our knowledge, this is the first time the notion of explicit confirmation is presented in the context of message recognition protocols.

Existing MRPs allow Alice to send a message to Bob and hope that he will accept this message as sent from Alice. By just following the MRP instructions, Alice does not have any means to know for sure wether or not Bob has accepted her message. *Explicit confirmation* provides such an assurance to Alice and, so far, it has not been considered in the context of MRPs.

In this context, we consider the usual adversarial goal, which is to make Bob accept a message that was not sent from Alice, as well as the added adversarial goal against the explicit confirmation which is to make Alice believe that Bob has accepted a message when, in fact, he has rejected, or vice versa.

Note that there is a separation between these two adversarial goals. Suppose Alice sends the message M. If Bob receives the message M', where $M \neq M'$, and he accepts it, then the adversary has achieved her first goal. On the other hand, suppose Alice sends M and Bob receives M' (possibly with $M = M'$). Now, for whatever reason, Bob rejects M'; however, Alice is made to believe that Bob has accepted in this session. In this case, the adversary has achieved her second goal.

A quick, and rather primitive, way of providing explicit confirmation is to have Bob authenticate a message to Alice, using an MRP, where the message is announcing the acceptance or rejection of the previous message from Alice. Note that the role of Alice and Bob is not symmetric and they cannot use the same hash chain pair $\{a_i\}$ and $\{b_i\}$ for both when Alice is sending messages and when Bob is sending messages; indeed, if they use the same hash chain, a man-in-the-middle attack is possible. Hence, they need to have two pairs of hash chains and basically get involved in a procedure which has almost the same complexity as the original primitive, in terms of memory, computation, and communication.

For the first time, we propose a new MRP which provides explicit confirmation without requiring any extra communication or computation between the participants. As for the memory, on the sender's side it has the same memory requirement as the plain MRP, however, on the receiver's side it has the same memory requirement of the aforementioned primitive solution, which is basically twice as much as the MRP without explicit confirmation.

7.1 A New Message Recognition Protocol with Explicit Confirmation

The main idea behind this protocol is for Bob to use two hash chains instead of one, $\{c_i\}$ for *confirming* the acceptance of messages and $\{d_i\}$ for *declining* the acceptance of messages. This of course will impact the verification conditions on Alice's side; however, the main core of the protocol remains intact. Alice uses one hash chain $\{a_i\}$ for sending her messages. In session i of the protocol, she is trying to authenticate the message m_i using key a_i. Bob authenticates himself to Alice, by sending c_i, to confirm that he has accepted the message from the previous session, or by sending d_i, if he has declined the message he received in the previous session. When receiving either c_i or d_i, Alice can check if Bob is the same person she was speaking to during the prior sessions and receive an explicit confirmation on wether or not her message was accepted by Bob.

As in previous sections, consider a one-way hash function $H : \{0,1\}^s \rightarrow \{0,1\}^s$ and a message authentication code $\mathrm{MAC} : \{0,1\}^s \times \{0,1\}^* \rightarrow \{0,1\}^c$, with typical parameters of $s \geq 80$ and $c \geq 30$. Fix n to be the maximum number of messages to be authenticated, or the maximum number of sessions.

Now, Alice randomly chooses a_0 and forms a hash chain of the form $a_i = H(a_{i-1})$, $i = 1, \ldots, n$. Alice uses a_i as keys for MAC values she computes in session i. Bob, on the other hand, randomly chooses c_0 and d_0. He forms $c_i = H(c_{i-1})$ and $d_i = H(d_{i-1})$, $i = 1, \ldots, n$. In particular, he uses c_i as keys when he accepts the message of the previous session, whereas he uses d_i as keys when he rejects the message of the previous session.

The initialization phase is constituted of Alice sending a_n to Bob over the narrow-band channel and Bob sending c_n and d_n to Alice over the same channel.

The internal state of Alice includes (along with each variable's initial value):

- $i_A := n - 1$: the position of Alice in her chain.
- $i_{confirm} := n$: the last index of Bob's chain corresponding to an accepted message by Bob.
- $c_A := c_n$: the last value of Bob's confirm chain that was accepted by Alice.
- $i_{decline} := n$: the last index of Bob's chain corresponding to a rejected message by Bob.
- $d_A := d_n$: the last value of Bob's decline chain that was accepted by Alice.
- $i_{acceptA} := n$: the last index of Bob's chain that was accepted by Alice.
- $b_A := c_n$: the last value of Bob's chain that was accepted by Alice.
- $M := Null$: the input message to be authenticated in the current session.
- a one-bit flag, to distinguish the program states **A0** and **A1**.

On the other hand, Bob's internal state is as follows:

- $i_B := n - 1$: the position of Bob in his chain.
- $i_{acceptB} := n$: the last index of Alice's chain that was accepted by Bob.
- $a_B := a_n$: the last value of Alice's chain that was accepted by Bob.
- $e' := Null$: the MAC value received in the current session, supposedly from Alice.
- $M' := Null$: the message received in the current session, supposedly from Alice.

- a one-trit flag, to distinguish the program states **B0**, **B1**, and **B2**.
- $r := 1$, a one-bit flag to distinguish when Bob accepts a message or not.

Alice and Bob start in program states **A0** and **B0**. We write commit-message(M, i_A) to indicate that Alice is committing herself to sending the message M to Bob in session i_A. Moreover, we write accepted-message(i_A) when Alice learns that Bob has accepted a message in session i_A. Similiarly, we write rejected-message(i_A) when Alice learns that Bob has rejected a message in session i_A. We let T be the maximum amount of time Alice waits to receive a response from Bob, and vice versa.

A0 is executed as follows:

```
If $i_A \leq 0$ then **Abort**.
Receive input $(M)$ and commit-message$(M, i_A)$.
Compute $e_{i_A} := \mathrm{MAC}_{a_{i_A}}(i_A \| M)$.
Send $[e_{i_A}, M]$ to Bob and **goto A1**.
```

B0 is executed as follows:

```
If $i_B \leq 0$ then **Abort**.
Wait to receive $[e', M']$, then **goto B1**.
```

B1 has the following description:

```
If $r = 1$ then $b := c_{i_B}$.
If $r = 0$ then $b := d_{i_B}$.
Send $[i_B, b]$ to Alice and **goto B2**.
```

A1 is performed in the following manner:

```
Wait at most time $T$ to receive $[i'_B, b']$.
If $[i'_B, b']$ is received, then
    Compute $j_c := i_{confirm} - i_A$ and $j_d := i_{decline} - i_A$.
    If $i'_B = i_{acceptA}$ and $b_A = b'$ (Bob has not received the last flow of the previous session) then
        Let $N := Null$.
        Send $[i_{acceptA}, a_{i_{acceptA}}, N]$ and **goto A0**.
    If $i'_B = i_A$ and $c_A = H^{j_c}(b')$ (Bob has accepted the message of the previous session) then
        Let $N := M$.
        Send $[i_A, a_{i_A}, N]$ to Bob and accepted-message$(i_{acceptA})$.
        Let $i_{acceptA} := i'_B$, $b_A := b'$, $i_{confirm} := i'_B$, $c_A := b'$ and $i_A := i_A - 1$. (Alice updates her state.)
        **goto A0**.
    If $i'_B = i_A$ and $d_A = H^{j_d}(b')$ (Bob has rejected the message of the previous session) then
        Let $N := M$.
        Send $[i_A, a_{i_A}, N]$ to Bob and rejected-message$(i_{acceptA})$.
        Let $i_{acceptA} := i'_B$, $b_A := b'$, $i_{decline} := i'_B$, $d_A := b'$ and $i_A := i_A - 1$. (Alice updates her state.)
        **goto A0**.
    else Resend $[e_{i_A}, M]$ to Bob and **goto A1**.
If timeout then
    Resend $[e_{i_A}, M]$ to Bob and **goto A1**.
```

B2 is performed as follows:

```
Wait at most time $T$ to receive $[i'_A, a', N']$.
If $[i'_A, a', N']$ is received, then
    If $i'_A = i_B$ and $a_B = H(a')$ then (Alice and Bob seem to be synchronized.)
        If $N' = M'$ and $e' = \mathrm{MAC}_{a'}(i'_A \| M')$ then
            Accept$(M', i_B)$ and let $r := 1$.
        else Accept$(Null)$ and let $r := 0$.
        Let $i_{acceptB} := i'_A$, $a_B := a'$ and $i_B := i_B - 1$. (Bob updates his state.)
        **goto B0**.
    else **goto B1**.
If timeout, then **goto B1**.
```

Figure 18 illustrates the common case of this protocol.

7.2 Analysis of Our New Message Recognition Protocol with Explicit Confirmation

In this section, we analyze our MRP with explicit confirmation and argue that under the assumptions described in Definitions 1, 2, and 3, it provides security against both adversarial goals—the usual adversarial goal against an MRP, which is to make Bob accept a message that was not sent from Alice, and the adversarial goal against the explicit confirmation which is to make Alice believe that Bob has accepted a message when he has in fact rejected, or vice versa.

Let us begin by considering an adversary who tries to deceive Bob into accepting a message that Alice did not send. We first look at the change in the verifying conditions. In this protocol, Bob verifies Alice and her messages the same way he did in the protocol of Section 5. However, the instructions for Alice to verify Bob have changed. Previously, Alice was supposed to check wether or not $b_A = H(b')$, where b_A is the last value of Bob's hash chain accepted by Alice and b' is what she received, supposedly from Bob, in this session. Now, Alice keeps two values c_A and d_A, the last values, respectively, from Bob's confirm and decline hash chains accepted by Alice. She also has two local parameters j_c and j_d that she computes in program state

```
Alice

Internal state: $i_A$, $i_{acceptA}$, $b_A$, $M$
$i_{confirm}$ and $c_A$, $i_{decline}$ and $d_A$.
A0:
    If $i_A \leq 0$ then Abort.
    Receive ($M$) and commit-message($M, i_A$).
    Compute $e_{i_A} := \mathrm{MAC}_{a_{i_A}}(i_A \| M)$.

    Send $[e_{i_A}, M]$.

A1:

    Receive $[i'_B, b']$.
    Compute $j_c := i_{confirm} - i_A$ and
    $j_d := i_{decline} - i_A$.
    If $i'_B = i_{acceptA}$ and $b_A = b'$ then
        Let $N := Null$.
        Send $[i_{acceptA}, a_{i_{acceptA}}, N]$ and goto A0.
    If $i'_B = i_A$ and $c_A = H^{j_c}(b')$ then
        Let $N := M$.

        Send $[i_A, a_{i_A}, N]$ and accepted-message($i_{acceptA}$).

        Let $i_{acceptA} := i'_B$, $b_A := b'$, $i_{confirm} := i'_B$,
        $c_A := b'$ and $i_A := i_A - 1$.
        goto A0.
    If $i'_B = i_A$ and $d_A = H^{j_d}(b')$ then
        Let $N := M$.
        Send $[i_A, a_{i_A}, N]$ and rejected-message($i_{acceptA}$).
        Let $i_{acceptA} := i'_B$, $b_A := b'$, $i_{decline} := i'_B$,
        $d_A := b'$ and $i_A := i_A - 1$.
        goto A0.
    else Resend $[e_{i_A}, M]$ and goto A1.

Messages

$\xrightarrow{e_{i_A}, M}$
$\xleftarrow{i_B, b}$
$\xrightarrow{i_A, a_{i_A}, N}$

Bob

Internal state: $i_B$, $i_{acceptB}$, $a_B$, $e'$, $M'$, $r$

B0:
    If $i_B \leq 0$ then Abort.

    Receive $[e', M']$.

B1:
    If $r = 1$ then $b := c_{i_B}$.
    If $r = 0$ then $b := d_{i_B}$.

    Send $[i_B, b]$.

B2:

    Receive $[i'_A, a', N']$.

    If $i'_A = i_B$ and $a_B = H(a')$ then
        If $N' = M'$ and $e' = \mathrm{MAC}_{a'}(i'_A \| M')$ then
            Accept($M'$, $i_B$) and let $r := 1$.
        else Accept($Null$) and let $r := 0$.
        Let $i_{acceptB} := i'_A$, $a_B := a'$ and $i_B := i_B - 1$.
        goto B0.
    else goto B1.
```

Figure 18: Our Proposed Message Recognition Protocol with Explicit Confirmation (Common Case)

A1. Alice first checks to see whether $c_A = H^{j_c}(b')$ holds. If so, Bob is verified and Alice is convinced that Bob has accepted her message in the previous session. Otherwise, she checks $d_A = H^{j_d}(b')$. If this latter equation holds, Bob is verified and Alice believes that her message from the previous session was not accepted by Bob.

Note that if either equation holds, Bob is verified. Moreover, note that j_c and j_d are defined in a way that they will always be greater than or equal to 1. Hence, finding a b' that makes either equation hold is no harder than finding a b' that verifies in $c_A = H(b')$ or $d_A = H(b')$. This brings us back to using the same security assumptions described in Definitions 1, 2, and 3. Moreover, attacks against this protocol reduce to attacks to our MRP of Section 6. Hence, any adversary against this protocol who tries to deceive Bob into accepting a message that Alice never sent is reduced to an adversary against the MRP of Section 5.

We next consider an adversary who tries to make Alice believe that Bob has accepted in the previous session, when, in fact, he has rejected. Such an adversary must convince Alice that first of all she is speaking to Bob; that is, b' is verified by Alice. When Bob rejects, he sets $b = d_{i_B}$. Now, Eve wants $c_A = H^{j_c}(b')$ to verify. Not having seen c_{i_B}, Eve wants to find a b' such that $H^{j_c}(c_{i_B}) = H^{j_c}(b')$. If she can efficiently find such a b', then she is reduced to an adversary who can attack the protocol of Section 5 by finding $b'' = H^{j_c - 1}(b')$ which hashes to $c_{i_B + j_c}$.

Similarly, an adversary who tries to make Alice believe that Bob has rejected in the previous session, when, in fact, he has accepted, also reduces to an adversary against the protocol of Section 5.

7.3 Security and Explicit Confirmation Theorem

We now sum up the above discussion into the following theorem which concludes the analysis of the security of the proposed message recognition protocol with explicit confirmation.

Theorem 4 *Consider an adversary against the protocol of Section 7.1 who either efficiently deceives Bob into accepting (M',i), where M' is not Null and Alice did not send M' in session i, or efficiently deceives Alice in believing that Bob has accepted a message when, in fact, he has rejected, or vice versa. Such an adversary implies an efficient algorithm that finds depth-i preimages or depth-i second preimages, or creates depth-i existential forgeries.*

8 Comments and Conclusion

We briefly reviewed the definitions and the security model of message recognition protocols in the literature. Then, we noted the equivalence of digital signature schemes and stateless non-interactive message recognition protocols. This equivalence suggests that interactive message recognition protocols are more appropriate and poses an open problem as to whether adding state to non-interactive MRPs can make them, as interactive MRPs are, more efficient than digital signature schemes.

We looked at the Jane Doe message recognition protocol proposed by Lucks et al. (2005) in more detail and described its inability to recover in case of a certain adversarial disruption. In particular, in case of communication failure or adver- sarial disruption, this protocol is not equipped with a practical resynchronization process and can fail to resume.

In order to overcome the recoverability problem of the Jane Doe protocol, we first suggested a variant of this protocol to overcome this problem which is equipped with a resynchronization technique that allows users to resynchronize whenever they wish or when they suspect an intrusion. This approach is simple but fixes the recoverability issue of the Jane Doe protocol at the price of having to call upon a separate procedure.

We further proposed a new message recognition protocol, which is based on the Jane Doe protocol, but this time, incorporates a resynchronization technique within itself and, hence, provides self-recoverability. We formally proved the security of our protocol.

It should be noted that our second solution is somewhat less efficient than the Jane Doe protocol in that each message M is transmitted twice (in the first flow, and again in the third flow of Figure 11). This would not be a problem if the communication channel is inexpensive. However, it (roughly) doubles the power consumption as compared to the Jane Doe protocol if messages are large. If this creates a problem, it would be possible to modify our protocol by sending $N = H(M)$ in the third flow instead of $N = M$. Then Bob checks that $N' = H(M')$ instead of $N' = M'$.

Finally, we proposed another message recognition protocol that provides explicit confirmation with the minimal effort of using an extra hash chain for Bob, and with no extra communication requirement.

Acknowledgements

We would like to thank Natural Sciences and Engineering Research Council of Canada (NSERC) and Mathematics of Information Technology and Complex Systems (MITACS) for supporting this research, and the anonymous reviewers for helping us to improve this paper.

REFERENCES

Anderson, R., Bergadano, F., Crispo, B., Lee, J.-H., Manifavas, C. & Needham, R. (1998), A New Family of Authentication Protocols, *in* ‘ACMOSR: ACM Operating Systems Review’, Vol. 32, pp. 9–20.

Gehrmann, C. (1998), ‘Multiround Unconditionally Secure Authentication’, *Designs, Codes, and Cryptography* **15**(1), 67–86.

Hammell, J., Weimerskirch, A., Girao, J. & Westhoff, D. (2005), Recognition in a Low-Power Environment, *in* ‘ICDCSW ’05: Proceedings of the Second International Workshop on Wireless Ad Hoc Networking (WWAN)’, IEEE Computer Society, Washington, DC, USA, pp. 933–938.

Lucks, S., Zenner, E., Weimerskirch, A. & Westhoff, D. (2005), Entity Recognition for Sensor Network Motes, *in* ‘GI Jahrestagung (2)’, pp. 145–149.

Lucks, S., Zenner, E., Weimerskirch, A. & Westhoff, D. (2008), Concrete Security for Entity Recognition: The Jane Doe Protocol, *in* ‘Progress in Cryptology—INDOCRYPT 2008’, Vol. 5365 of *Lecture Notes in Computer Science*, Springer, pp. 158–171.

Lucks, S., Zenner, E., Weimerskirch, A. & Westhoff, D. (2009), ‘Concrete Security for Entity Recognition: The Jane Doe Protocol (Full Paper)’, Cryptology ePrint Archive, Report 2009/175. `http://eprint.iacr.org/`.

Mashatan, A. & Stinson, D. R. (2008), A New Message Recognition Protocol for Ad Hoc Pervasive Networks, *in* M. K. Franklin, L. C. K. Hui & D. S. Wong, eds, ‘CANS’, Vol. 5339 of *Lecture Notes in Computer Science*, Springer, pp. 378–394.

Menezes, A., van Oorschot, P. C. & Vanstone, S. A. (1996), *Handbook of Applied Cryptography*, CRC Press.

Mitchell, C. J. (2003), Remote User Authentication Using Public Information, *in* K. G. Paterson, ed., ‘IMA Int. Conf.’, Vol. 2898 of *Lecture Notes in Computer Science*, Springer, pp. 360–369.

Weimerskirch, A. & Westhoff, D. (2003), Zero Common-Knowledge Authentication for Pervasive Networks, *in* ‘Selected Areas in Cryptography’, Vol. 3006 of *Lecture Notes in Computer Science*, Springer, pp. 73–87.