A NEW MESSAGE RECOGNITION PROTOCOL WITH SELF-RECOVERABILITY FOR AD HOC PERVASIVE NETWORKS

IAN GOLDBERG[1], ATEFEH MASHATAN[2], AND DOUGLAS R. STINSON[3]

Abstract. We examine the problem of message recognition by reviewing the definitions and the security model in the literature. In particular, we look at a protocol by Lucks et al. more closely and note its inability to recover in case of a certain adversarial disruption. This shortcoming can be fixed by employing a separate resynchronization process and this has already been done in the literature. However, it is also of interest to remedy this shortcoming without using a separate procedure. We propose a new message recognition protocol, which is based on the original protocol by Lucks et al., and incorporate the resynchronization technique within the protocol itself. That is, without having to provide a separate resynchronization procedure, we overcome the recoverability problem of the protocol. Moreover, we enumerate all possible attacks against the protocol and show that none of the attacks can occur. We further prove the security of the protocol and its ability to self-recover once the disruption has stopped.

1. Introduction

Entity recognition is a weaker security notion than entity authentication; it 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. There is an analogous correspondence between message recognition and message authentication.

There have been several recent papers on designing protocols where the source of trust is a narrow-band authenticated channel; see for example [3], [5], [8], [9], and [10]. In particular, there has been recent interest in designing recognition protocols using this communication model. This problem has been considered in a context where we are dealing with low-computational power devices 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.

Lucks et al. [3] motivated this model with the following example. Let Alice and Bob be two strangers who meet in a party for the first time. They leave the party after making a bet. Some

Date: October 9, 2008.

[1] David R. Cheriton School of Computer Science
iang@cs.uwaterloo.ca
[2] Department of Combinatorics and Optimization
amashatan@uwaterloo.ca
[3] David R. Cheriton School of Computer Science
dstinson@uwaterloo.ca
University of Waterloo
Waterloo, Ontario CANADA N2L 3G1

.

days later, it turns out that Alice wins the bet. Afterward, 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. Bob wants to be assured that this message is indeed sent from the entity who introduced herself as "Alice" in the party. In other words, Bob needs to *recognize* "Alice", whoever she is, or a message that is sent from her.

Now consider Alice and Bob to be two small devices who "meet" in a somewhat secure environment that allows them to send authenticated, but not confidential, messages. They are later placed in a hostile environment where Alice wants Bob to recognize the messages sent from her to Bob. An adversary, Eve, is present all along. When Alice and Bob first meet, Eve can read the authenticated messages, but cannot change them. Later, when Alice and Bob are placed in a hostile environment, Eve can not only read, but also modify messages. She can also insert her own messages claiming to be from either party. Eve's goal is to make Bob accept messages from her as sent from Alice, where Alice has never, or at least not recently, sent those messages.

Since message recognition is weaker than message authentication, every message authentication protocol trivially provides message recognition. Moreover, message recognition can be achieved using public-key, when public-key computations are feasible, or secret-key cryptography, when predeployed authentic information is available. However, in some scenarios, public-key computations may be too costly and there may be no secure channel where the secret keys can be transmitted confidentially.

One can ask what security goals can be achieved in such a constrained model? There are claims in the literature, see [10] for example, suggesting that achieving message authentication is not possible in such an environment. Hence, they pursue the weaker security of message recognition.

We look at a message recognition protocol proposed by Lucks et al. 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 inability to recover was noted previously [7] and fixed by calling upon a separate procedure called a "resynchronization protocol". We propose a message recognition protocol that is able to recover without having to call a separate protocol. That is, the proposed protocol has the advantage of self-recoverability. We formally prove that our protocol is secure and fully recovers once the disruptions have stopped.

The rest of the paper is organized as follows. Section 2 is devoted to examining previous recognition protocols and noting their shortcomings. In Section 3, we describe a new message recognition protocol. Finally, Section 4 is devoted to proving the security and recoverability of the protocol.

2. Previous Recognition Protocols

In this section, we briefly review the existing message recognition protocols and discuss their usability in the context of networks with low-computational power devices that also have low communication bandwidth.

There are two communication channels considered in the setting of recognition protocols: an insecure broadband channel, denoted by $\rightarrow$, and an authenticated non-confidential narrow-band

channel denoted by $\Rightarrow$. The broadband channel is available all the time and the narrow-band channel is only accessible once, for the initial session between two users.

Weimerskirch et al. [10] proposed a protocol called Zero Common-Knowledge (ZCK). This protocol is the starting point of a series of recent publications; see for example [2], [3], [4], [7], [6]. The ZCK protocol uses message authentication codes (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. The length of the hash chain, n, is fixed at the beginning and H is a one-way hash function.

Hammell et al. [2] 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, low energy resources. They concluded that it does exhibit these requirements, however, denial-of-service and memory complexity are areas of concern and needed to be addressed or improved upon in the future.

Hammell et al. did not investigate the security properties of the ZCK protocol, but rather relied on the security proof that came along with it. However, Lucks et al. [3] found a mistake in the security proof of this protocol and presented a practical attack against it. 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. For ease of reference, we will refer to the protocol proposed by Lucks et al. as the Lucks 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 considered as 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 Lucks 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 $\Rightarrow$.

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 Lucks protocol is given next.

Alice's internal state in the Lucks 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 Lucks protocol:

A0 (Alice's initial program state) Obtain m_i (possibly from Eve), then
  Commit-message(m_i, i).
  Compute $d_i = \text{MAC}_{a_i}(m_i)$.
  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**.

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 $\text{MAC}_{a'}(m_i) = d_i$ 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**.

Lucks et al. present their protocol in an extended abstract [3], and prove its security in the full version of the paper [4]. The protocol is proved to be secure given that the preimage resistance, second preimage resistance, and unforgeability properties, and their hash chain equivalents, hold. These properties are described in Section 4.

Although the Lucks 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. This particular disruption was previously noted in [7].

Figure 1 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 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. 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.

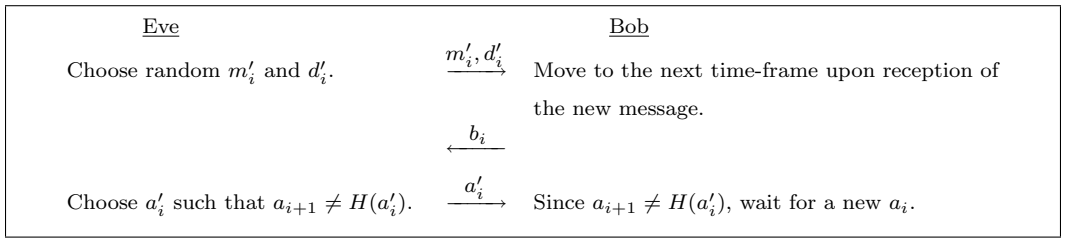

FIGURE 1. Eve "trapping" Bob in state **B1**

Eve can play the same trick with Alice and trap her in program state **A1** for an indeterminate period of time; Figure 2 illustrates this situation.

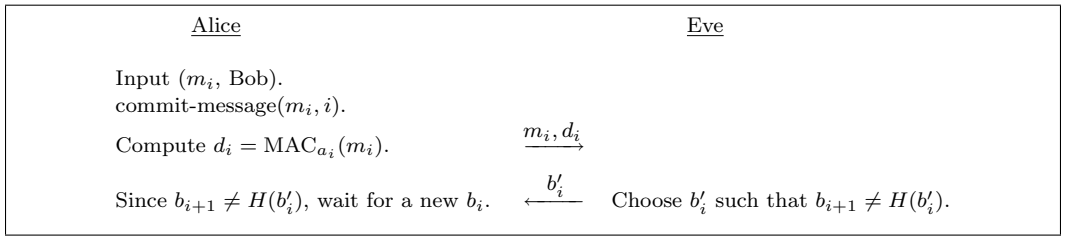

FIGURE 2. Eve "trapping" Alice in state **A1**

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.

As was mentioned before, [7] first noted these disruptions and proposed an improved version of the protocol which is equipped with a separate resynchronization protocol to fix this problem. 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.

Next, we propose a message recognition protocol which attains self-recoverability in case of the noted disruptions.

3. A New Message Recognition Protocol

We describe the details of our proposed recognition protocol in this section, while the security and recoverability analyses are postponed to the next session. Although this protocol is based on the original 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 protocol. However, 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 Lucks 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 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**.

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 $i_A \leq 0$ 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:

> 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 3 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.

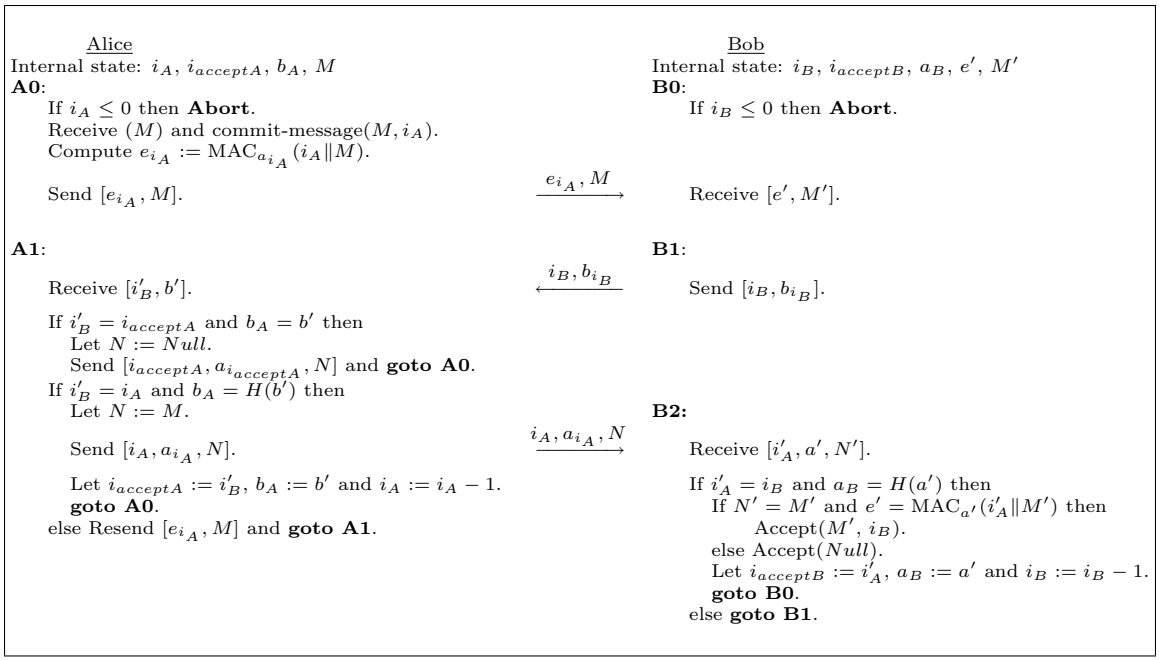

FIGURE 3. Our Proposed Message Recognition Protocol (Common Case)

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.

4. Security of Our New Message Recognition Protocol

In this section, we begin by listing the required security properties of the hash function H and the message authentication code MAC. Then, we consider different types of possible attacks against our protocol. Finally, we conclude with a theorem which ensures the security of our protocol.

4.1. **Security Assumptions.**

In this section, we list the security assumptions required for this protocol. These definitions are notions defined by Lucks et al. [3].

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.*

4.2. **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 [1] 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}.
- **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 [1]. These attacks are denoted AABB, ABBA, BABA, ABAB, BBAA, and BAAB. We will look at these different attacks separately. We stress that [1] 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. We outline these three reductions in the Appendix. 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.

4.2.1. *Attack of Type AABB.*

Figure 4 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.

4.2.2. *Attack of Type BBAA.*

Figure 5 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

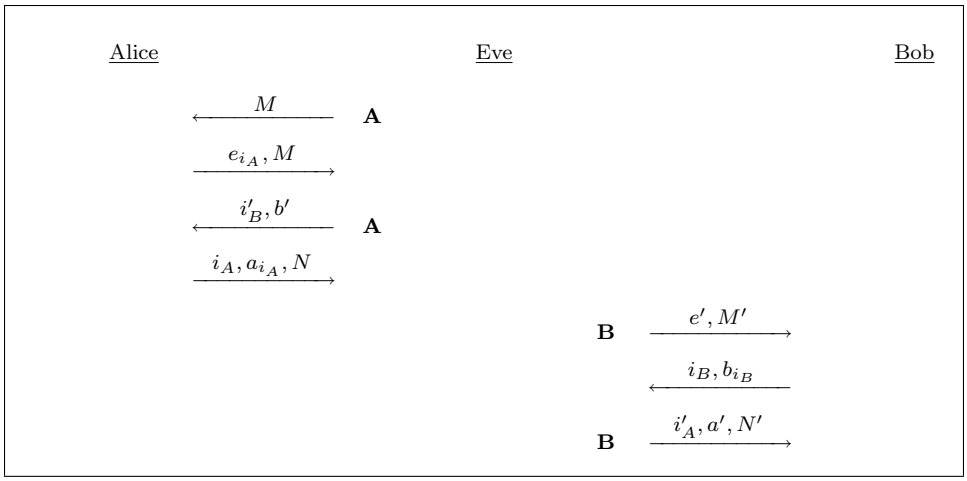

FIGURE 4. Attack of Type AABB

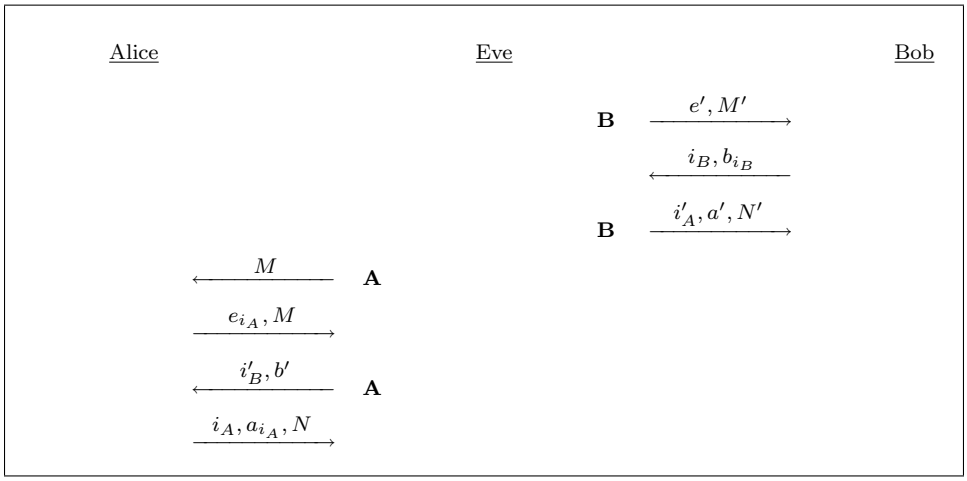

FIGURE 5. Attack of Type BBAA

$a_B = a_{i+1}$, 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.

4.2.3. *Attack of Type ABAB.*

Depicted in Fig. 6 is the ABAB attack.

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.

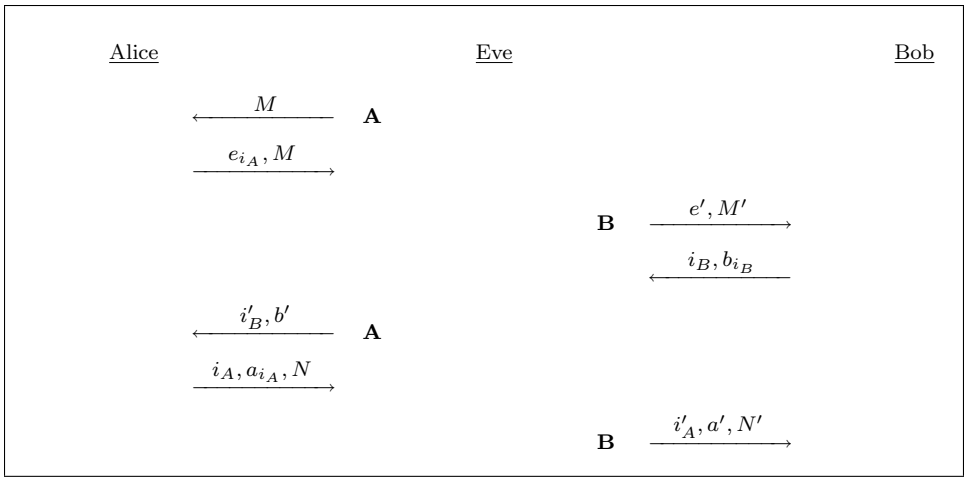

FIGURE 6. Attack of Type ABAB

4.3. **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 4.2.

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.

4.4. **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 Lucks 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.

4.5. **Main 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.

Security and Self-recoverability Theorem. *A successful adversary against the protocol of Section 3 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.*

5. Conclusion

We briefly reviewed the definitions and the security model of message recognition protocols in the literature. We looked at the message recognition protocol proposed by Lucks et al. [3] in more detail and described its inability to recover in case of a certain adversarial disruption, previously noted in the literature. We proposed a new message recognition protocol, which is based on the original protocol by Lucks et al., and which incorporates a resynchronization technique within itself to provide self-recoverability. That is, the proposed protocol overcomes the recoverability problem of the Lucks et al. protocol without having to provide a separate resynchronization procedure. Finally, we formally proved the security of our protocol.

References

[1] Christian Gehrmann. Multiround unconditionally secure authentication. *Designs, Codes, and Cryptography*, 15(1):67–86, 1998.

[2] Jonathan Hammell, André Weimerskirch, Joao Girao, and Dirk Westhoff. Recognition in a low-power environment. In *ICDCSW '05: Proceedings of the Second International Workshop on Wireless Ad Hoc Networking (WWAN) ICDCSW'05)*, pages 933–938, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Stefan Lucks, Erik Zenner, André Weimerskirch, and Dirk Westhoff. Entity recognition for sensor network motes. In *GI Jahrestagung (2)*, pages 145–149, 2005.

[4] Stefan Lucks, Erik Zenner, André Weimerskirch, and Dirk Westhoff. Is this Message From Alice? Efficient and Secure Entity Recognition for Low-End Devices, 2007. Manuscript.

[5] Atefeh Mashatan and Douglas R. Stinson. Interactive two-channel message authentication based on interactive-collision resistant hash functions. Technical Report 2007-24, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2007.

[6] Atefeh Mashatan and Douglas R. Stinson. A new message recognition protocol for ad hoc pervasive networks. Technical Report 2008-15, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2008.

[7] Atefeh Mashatan and Douglas R. Stinson. Recognition in ad hoc pervasive networks. Technical Report 2008-12, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2008.

[8] Chris J. Mitchell. Remote user authentication using public information. In Kenneth G. Paterson, editor, *IMA Int. Conf.*, volume 2898 of *Lecture Notes in Computer Science*, pages 360–369. Springer, 2003.

[9] Sylvain Pasini and Serge Vaudenay. An optimal non-interactive message authentication protocol. In David Pointcheval, editor, *Topics in Cryptography*, volume 3860 of *Lecture Notes in Computer Science*, pages 280–294, San Jose, California, U.S.A., February 2006. Springer-Verlag.

[10] André Weimerskirch and Dirk Westhoff. Zero common-knowledge authentication for pervasive networks. In *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2003.

Appendix A. Reducing Three Single-session Attacks

As was described in Section 4, Gehrmann [1] formally proves that there are only six possible types of single-session attack against the protocol of Figure 3. We analyzed the AABB, BBAA, and ABAB attacks in that section. Here we examine the remaining three attacks: BABA, BAAB, and ABBA. The BABA attack is reduced to the ABBA attack. Then, the ABBA attack is reduced to the ABAB attack. Finally, the BAAB attack is also reduced to the ABAB attack. This concludes the analysis of the six different attack scenarios.

A.1. **Reducing the BABA attack to an ABBA attack.**

The ABBA attack scenario, depicted in Fig. 7, 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.

On the other hand, the BABA attack scenario, illustrated in Fig. 8, 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.

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

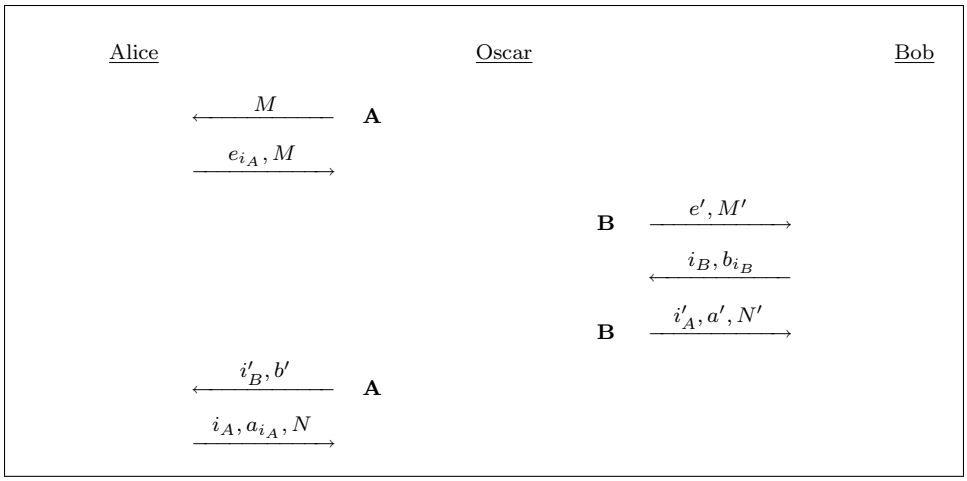

FIGURE 7. Attack of Type ABBA

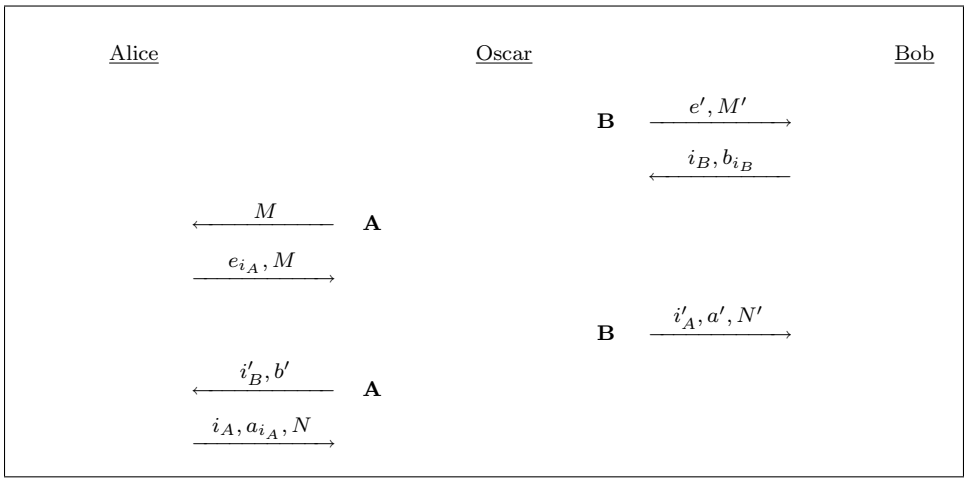

FIGURE 8. Attack of Type BABA

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.

A.2. **Reducing the ABBA attack to an ABAB attack.**

Recall the ABAB attack scenario from Section 4:

- **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.

A.3. **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 9 depicts this attack.

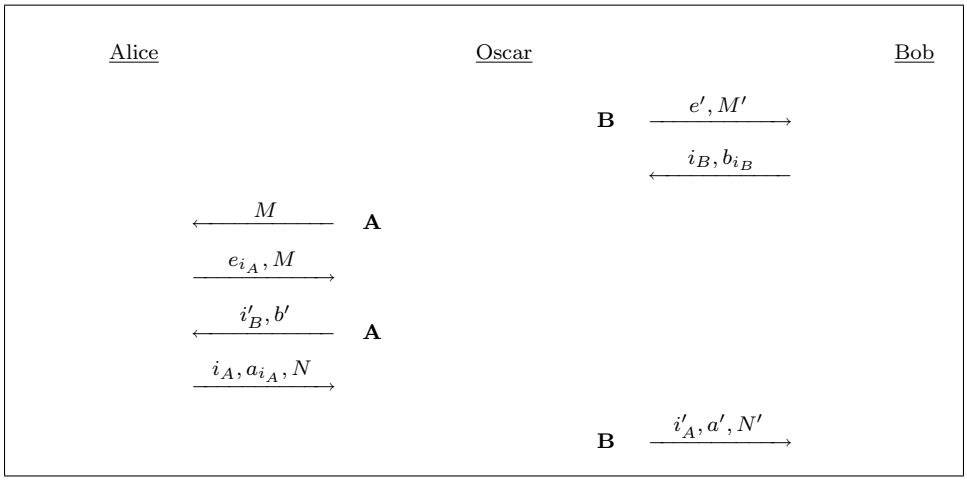

FIGURE 9. Attack of Type BAAB

The analysis of this case is analogous to that of Section A.1. 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.