Cryptanalysis of a Message Recognition Protocol by Mashatan and Stinson

Madeline Gonzalez and Rainer Steinwandt

Department of Mathematical Sciences, Florida Atlantic University,
777 Glades Road, Boca Raton, FL 33431, USA,
{mgonza29,rsteinwa}@fau.edu

Abstract. At CANS 2008, Mashatan and Stinson suggested a message recognition protocol for ad hoc pervasive networks. The protocol provides a procedure to resynchronize in case of a (possibly adversarial) disruption of communication. We show that this resynchronization process does not provide the functionality intended and in fact enables an adversary to create selective forgeries. The computational effort for the attack is negligible and allows the insertion of arbitrary messages.

Keywords: cryptanalysis, message recognition, ad hoc network

1 Introduction

In [MS08], Mashatan and Stinson propose *a new message recognition protocol for ad hoc pervasive networks*, aiming at scenarios with resource restricted devices. Their protocol relies on the use of a cryptographic hash function providing suitable guarantees, and the protocol avoids the use of asymmetric cryptography. In informal terms, the scenario in [MS08] can be summarized as follows: during an initialization phase, two parties A and B are connected through an authentic channel of low bandwidth. While this narrow-band channel can be eavesdropped, the adversary is confined to be *passive*, i. e., no messages can be altered, deleted or inserted. Later on, A and B are connected via a public broadband channel that is completely controlled by the, now *active*, adversary. The protocol in [MS08] tries to make sure that messages sent over this public insecure channel by A are only accepted by B if they indeed originate from the party A with which the initialization phase was performed. Further, according to [MS08], the proposed protocol *provides a practical procedure for resynchronization in case of any adversarial disruption or communication failure*

Mashatan and Stinson's proposal can be be seen in the same line of research as, for instance, Anderson et al.'s *Guy Fawkes protocol* [ABC+98], Stajano and Anderson's *resurrecting duckling* [SA00], Mitchell's scheme

for remote user authentication [Mit03], Weimerskirch and Westhoff's *zero common-knowledge authentication* [WW04], and Lucks et al.'s *Jane Doe protocol* [LZWW08].

Our contribution. Below we show that the resynchronization mechanism suggested by Mashatan and Stinson unfortunately does not work as intended, but actually enables an attack: an adversary can abuse the resynchronization process to send forged messages that are accepted as legitimate. The computational effort for the attack is negligible, and there is no restriction on the contents of the messages that can be inserted.

2 The proposal from CANS 2008

This section recalls Mashatan and Stinson's proposal from CANS 2008 to the extent necessary for describing our attack. The protocol splits into three components, which we discuss in the subsequent three subsections. For more details we refer to the original paper [MS08], which elaborates on the underlying assumptions on the hash function H (*pre-image resistance*, *paired second pre-image resistance*, *paired collision resistance*, *binding pre-image resistance*), for instance. We denote *passwords*[1] for party A by x_i and for party B by y_i. Writing H for the underlying hash function, we set $X_i := H(x_i)$, $Y_i := H(y_i)$ and refer to the X_i and Y_i as *committing hash values* of the passwords. Finally, the *binding hash values* are denoted by $\mathcal{X}_{i(i+1)}$ and $\mathcal{Y}_{i(i+1)}$ for A and B respectively, where $\mathcal{X}_{i(i+1)} := H(x_i, X_{i+1})$ and $\mathcal{Y}_{i(i+1)} := H(y_i, Y_{i+1})$.

At any given time, the internal state of A is given by an 8-tuple $(x_i, x_{i+1}, X_i, X_{i+1}, \mathcal{X}_{i(i+1)}, y^*_{i-1}, Y^*_i, \mathcal{Y}^*_{i(i+1)})$ with y^*_{i-1}, Y^*_i, $\mathcal{Y}^*_{i(i+1)}$ being B's most recent password, committing hash value, and binding hash value accepted by A. Likewise, the internal state of B is given by an 8-tuple $(y_i, y_{i+1}, Y_i, Y_{i+1}, \mathcal{Y}_{i(i+1)}, x^*_{i-1}, X^*_i, \mathcal{X}^*_{i(i+1)})$ with x^*_{i-1}, X^*_i, $\mathcal{X}^*_{i(i+1)}$ being A's most recent password, committing hash value, and binding hash value accepted by B.

Adversarial model During the initialization phase of the protocol, the involved parties A and B exchange information through an authenticated channel which we will denote by $\Longrightarrow$. The adversary is restricted to passive eavesdropping of this channel, no delaying, deleting, inserting, or

[1] Here we follow the terminology in [MS08] and stress that exhausting all possible passwords is assumed to be infeasible. In particular, this use of the term *password* differs from the common use in the context of password authenticated key establishment.

altering of messages is allowed. During the execution of the protocol and in the resynchronization process, A and B communicate over an insecure channel which we denote by $\longrightarrow$. The adversary has full control over the insecure channel, and in particular can delete and insert messages. The goal of the adversary is to create a forgery, i. e., to provoke a situation where B accepts a message-recipient pair (A, m) where the message m has never been sent by A.

2.1 Initialization phase

Figure 1 shows the steps performed by A and B in the initialization phase.

$\boldsymbol{A}$		$\boldsymbol{B}$
Choose random x_0 and x_1 and form $X_0 := H(x_0)$, $X_1 := H(x_1)$, and $\mathcal{X}_{01} := H(x_0, X_1)$.	$\xRightarrow{X_0, \mathcal{X}_{01}}$	Receive $X_0, \mathcal{X}_{01}$.
Receive $Y_0, \mathcal{Y}_{01}$.	$\xLeftarrow{Y_0, \mathcal{Y}_{01}}$	Choose random y_0 and y_1 and form $Y_0 := H(y_0)$, $Y_1 := H(y_1)$, and $\mathcal{Y}_{01} := H(y_0, Y_1)$.
Let $y^*_{-1} := \perp$, so A's initial state is $(x_0, x_1, X_0, X_1, X_{01}, \perp, Y_0, \mathcal{Y}_{01})$.		Let $x^*_{-1} := \perp$ so B's initial state is $(y_0, y_1, Y_0, Y_1, Y_{01}, \perp, X_0, \mathcal{X}_{01})$

Fig. 1. Initialization phase of [MS08]

In summary, during the initialization phase, A does the following:

- Choose random x_0 and x_1.
- Compute $X_0 := H(x_0)$, $X_1 := H(x_1)$, and $\mathcal{X}_{01} := H(x_0, X_1)$.
- Send X_0, $\mathcal{X}_{01}$ to B over the authenticated channel.
- Receive Y_0, and $\mathcal{Y}_{01}$ from B over the authenticated channel.
- Set $y^*_{-1} := \perp$, $Y^*_0 := Y_0$, $\mathcal{Y}^*_0 := \mathcal{Y}_0$.

Similarly, B performs the following steps:

- Choose random y_0 and y_1.
- Compute $Y_0 := H(y_0)$, $Y_1 := H(y_1)$, and $\mathcal{Y}_{01} := H(y_0, Y_1)$.
- Send Y_0, $\mathcal{Y}_{01}$ to A over the authenticated channel.
- Receive X_0, and $\mathcal{X}_{01}$ from A over the authenticated channel.

– Set $x^*_{-1} := \perp$, $X^*_0 := X_0$, $\mathcal{X}^*_0 := \mathcal{X}_0$.

The values X_0, $\mathcal{X}_{01}$, Y_0, $\mathcal{Y}_{01}$ which are interchanged by A and B over the authenticated channel can be eavesdropped—but not altered—by the adversary.

2.2 Execution of the protocol

Once the initialization phase has been completed, the actual protocol execution can take place as described in Figure 2.

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, y^*_{-1}, Y^*_0, \mathcal{Y}^*_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, x^*_{-1}, X^*_0, \mathcal{X}^*_{01})$
A		***B***
Receive input (m, B). Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$. Compute $h := H(m, x_0)$.	$\xrightarrow{m,h}$	Receive m', h'.
Receive y'_0, Y'_1, $\mathcal{Y}'_{12}$.	$\xleftarrow{y_0,Y_1,\mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
If $H(y'_0) = Y^*_0$ and $H(y'_0, Y'_1) = \mathcal{Y}^*_{01}$, then send x_0, X_1, $\mathcal{X}_{12}$ and update your internal state as follows: $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$ else initiate resynchronization.	$\xrightarrow{x_0,X_1,\mathcal{X}_{12}}$	Receive x'_0, X'_1, $\mathcal{X}'_{12}$. If $H(x'_0) = X^*_0$, $H(x'_0, X'_1) = \mathcal{X}^*_{01}$, and $h' = H(m', x')$, then update your internal state as follows: $(y_1, y_2, Y_1, Y_2, \mathcal{Y}_{12}, x'_0, X'_1, \mathcal{X}'_{12})$ and output (A, m') else initiate resynchronization.

Fig. 2. Protocol execution of [MS08]

Summarizing, on input a message-recipient pair (m, B), A does the following during a protocol execution:

– Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$.
– Compute $h := H(m, x_0)$.
– Send (m, h) and wait to receive y'_0, Y'_1, $\mathcal{Y}'_{01}$ from B. Resend if B does not respond.
– If $H(y'_0) = Y^*_0$ and $H(y'_0, Y'_1) = \mathcal{Y}_{01}$, send x_0, X_1, $\mathcal{X}_{01}$ to B and update the internal state to $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$, else initiate resynchronization.

After receiving (m', h'), B does the following:

- Choose a random y_2 and compute $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
- Send y_0, Y_1, $\mathcal{Y}_{12}$ to A and wait to receive x'_0, X'_1, $\mathcal{X}_{12}$. Resend if A does not respond.
- If $H(x'_0) = X^*_0$, $H(x'_0, X'_1) = \mathcal{X}_{01}$, and $h' = H(m', x'_0)$ then update the internal state to $(y_1, y_2, Y_1, Y_2, \mathcal{Y}_{12}, x'_0, X'_1, \mathcal{X}'_{12})$ and output (A, m'), else initiate resynchronization.

Note that all messages are sent over an insecure channel, where the adversary can delete, modify, and insert messages at will. Further, it is possible for A to update its internal state after sending x_0, X_1, $\mathcal{X}_{12}$ without B updating its state. Therefore, the resynchronization process that follows is not symmetric.

2.3 Resynchronization process

In the case of adversarial intrusion or communication failure, either A or B can initiate the resynchronization process in Figure 3. As shown in this figure, B has two sets of conditions that can update its internal state, whereas A has only one.

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, y^*_{-1}, Y^*_0, \mathcal{Y}^*_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, x^*_{-1}, X^*_0, \mathcal{X}^*_{01})$
A		***B***
Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$.		Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Receive y'_0, Y'_1, $\mathcal{Y}'_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Send y_0, Y_1, $\mathcal{Y}_{12}$.
Send x_0, X_1, $\mathcal{X}_{12}$.	$\xrightarrow{x_0, X_1, \mathcal{X}_{12}}$	Receive x'_0, X'_1, $\mathcal{X}'_{12}$.
If $y^*_{-1} = y'_0$ and $Y^*_0 = Y'_1$, then $\mathcal{Y}^*_{01} := \mathcal{Y}'_{12}$, else initiate resynchronization.		If $x^*_{-1} = x'_0$ and $X^*_0 = X'_1$, then $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$, otherwise if $H(x'_0) = X^*_0$ and $H(x'_0, X'_1) = \mathcal{X}^*_{01}$, then $x^*_{-1} := x'_0$, $X^*_0 := X'_1$, $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$ else initiate resynchronization.

Fig. 3. Resynchronization process of [MS08]

We can summarize the resynchronization process as follows:

- A and B respectively choose random x_2, y_2 and form $X_2 := H(x_2)$, $Y_2 := H(y_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$, and $\mathcal{Y}_{12} := H(y_1, Y_2)$.
- B sends y_0, Y_1, $\mathcal{Y}_{01}$ to A.
- A sends x_0, X_1, $\mathcal{X}_{01}$ to B.
- If $y^*_{-1} = y'_0$ and $Y^*_0 = Y'_1$, then A sets $\mathcal{Y}^*_{01} := \mathcal{Y}'_{12}$, else initiates resynchronization.
- If $x^*_{-1} = x'_0$ and $X^*_0 = X'_1$, then B sets $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$, else if $H(x'_0) = X^*_0$ and $H(x'_0, X'_1) = \mathcal{X}^*_{01}$, then B sets $x^*_{-1} := x'_0$, $X^*_0 := X'_1$, $\mathcal{X}^*_{01} := \mathcal{X}'_{12}$, else initiates resynchronization.

During resynchronization, A can only refresh the value $\mathcal{Y}^*_{01}$, whereas B can either refresh the value $\mathcal{X}^*_{01}$ or update x^*_{-1}, X^*_0, $\mathcal{X}^*_{01}$.

3 Provoking an unrecoverable situation

If A or B suspects a communication failure or a possible adversarial intrusion, it can initiate the resynchronization process. Here we show that

- an adversary can create a situation where A keeps on initiating the resynchronization process, but the protocol does not recover, and
- an adversary can create a situation where B keeps on initiatiating the resynchronization process, but the protocol does not recover.

It is worth noting that in both cases, modification of a single message on the public channel is sufficient, i. e., the adversary does not have to stay "online" for achieving this type of denial of service: these attacks are qualitatively different from simply blocking communication between A and B. Section 4 builds on these observations to create a successful forgery.

3.1 Unrecoverability with resynchronization initiated by *A*

As depicted in Figure 4, assume that after a successful initialization phase, A has internal state $(x_0, x_1, X_0, X_1, X_{01}, \perp, Y_0, \mathcal{Y}_{01})$ and B has internal state $(y_0, y_1, Y_0, Y_1, Y_{01}, \perp, X_0, \mathcal{X}_{01})$. Now A starts executing a protocol as specified in Section 2.2, sending a message m along with matching h-value to B. In response to this, B sends y_0, Y_1 and $\mathcal{Y}_{12}$.

The adversary can replace y_0 with a (random) value such that A's validity check $H(y'_0) = Y_0$ and $H(y'_0, Y'_1) = \mathcal{Y}^*_{01}$ fails. Following the protocol specification, now A initiates the resynchronization process (see upper

part of Figure 4). Note that so far A never updated its internal state and still has stored the values $y^*_{-1} = \perp$, $Y_0* = Y_0$, and $\mathcal{Y}^*_{01} = \mathcal{Y}_{01}$.

Now, in the resynchronization phase, B sends to A the values y'_0, Y'_1, and $\mathcal{Y}'_{12}$. These values do not match the values stored by A, however. Consequently, A initiates resynchronization again. Re-running the resynchronization will not help the situation, so the protocol becomes unrecoverable. Figure 4 summarizes the sequence of events.

3.2 Unrecoverability with resynchronization initiated by B

Consider a second scenario as in Figure 5. Assume that after a successful initialization phase A has internal state $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, \perp, Y_0, \mathcal{Y}_{01})$ and B has internal state $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$ as before. As before A initiates an execution of the protocol in [MS08] by sending a message m along with matching h-value to B. In response, A receives y'_0, Y'_1, and $\mathcal{Y}'_{12}$ from B. Our adversary faithfully transmits these messages, so that A's validity check succeeds and A updates its internal state to $(x_1, x_2, X_1, X_2, X_{12}, y'_0, Y'_1, \mathcal{Y}'_{12})$. Further, A sends x_0, X_1 and $\mathcal{X}_{12}$ to B. Our adversary can replace x_0 with a (random) value so that the values x'_0, X'_1, and $\mathcal{X}'_{12}$ received by B from A do not verify. Consequently, following the protocol specification in Section 2.2, B will initiate the resynchronization process. Note that so far B never updated its internal state and has stored $x^*_{-1} = \perp$, $X^*_0 = X_0$, and $\mathcal{X}^*_{01} = \mathcal{X}_{01}$.

In the resynchronization process, A sends x_1, X_2, and $\mathcal{X}_{23}$ to B. Even if the values x'_1, X'_2, and $\mathcal{X}'_{23}$ received by B are identical to the values sent by A, $x'_1 \neq x^*_{-1}$ and $H(x'_1) \neq X^*_0$ cause B to initiate resynchronization again. Re-running the resynchronization will not resolve the situation, and analogously as in the previous section the protocol becomes unrecoverable. Figure 5 summarizes the sequence of events.

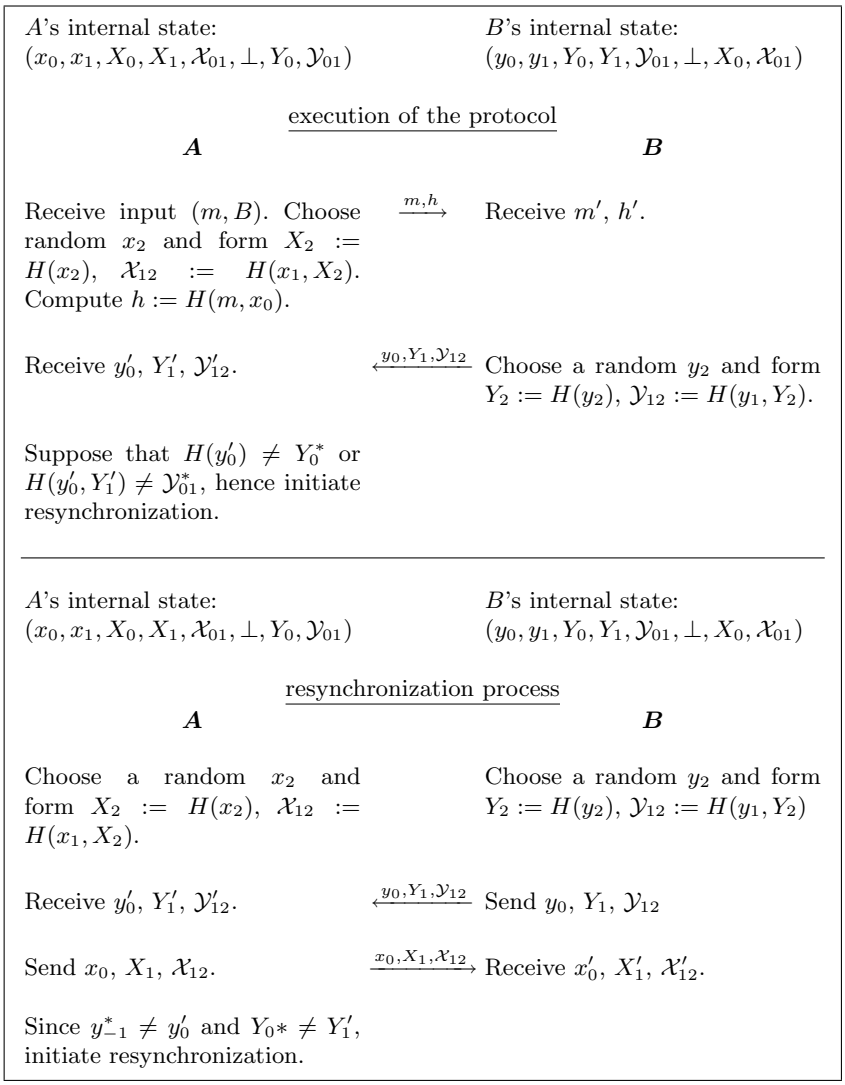

Fig. 4. Unrecoverability after a resynchronization initiated by A

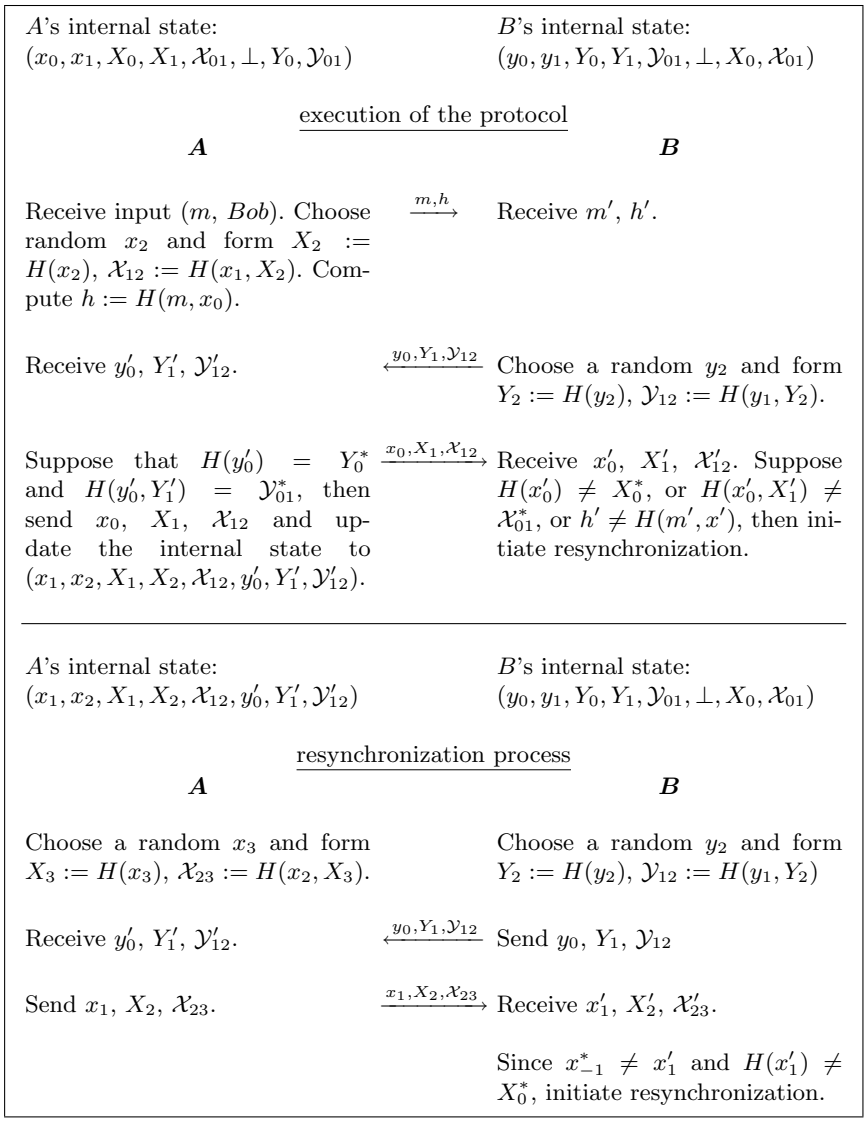

Fig. 5. Unrecoverability after a resynchronization initiated by B

4 Creating a forgery

To describe the attack, in subsequent figures we denote the adversary by F. Messages delivered faithfully by F are denoted by $\rightharpoonup$ and the messages created by F are denoted by $\rightharpoondown$. To begin our attack, we assume that A and B have successfully completed the initialization phase of the protocol. From here on, the attack unfolds in four steps:

1. executing the message recognition protocol
2. first resynchronization (unsuccessful)
3. second resynchronization (successful)
4. executing the message recognition protocol a second time

The subsequent four subsections elaborate on each of these steps.

4.1 Execution of the recognition protocol

In this first step, the goal of F is to learn the initial password x_0 from A. For this, F proceeds as shown in Figure 6.

A's internal state: $(x_0, x_1, X_0, X_1, \mathcal{X}_{01}, \perp, Y_0, \mathcal{Y}_{01})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$
A	***F***	***B***
Receive (m, B) as input. Choose a random x_2 and form $X_2 := H(x_2)$, $\mathcal{X}_{12} := H(x_1, X_2)$. Compute $h := H(m, x_0)$.	$\xrightarrow{m,h}$	Receive m, h.
Receive y_0, Y_1, $\mathcal{Y}_{12}$.	$\xleftarrow{y_0,Y_1,\mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(x_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Since $H(y_0) = Y_0^*$ and $H(y_0, Y_1) = \mathcal{Y}_{01}$, send x_0, X_1, $\mathcal{X}_{12}$ and update the internal state to $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$.	$\xrightarrow{\widetilde{x},X_1,\mathcal{X}_{12}}$	Since $H(\widetilde{x}) \neq X_0$ initiate resynchronization.

Fig. 6. First step of the attack: execution of the protocol

Summarizing, in this first step of the attack F does the following:

– Forward m, h faithfully from A to B.
– Forward the values y_0, Y_1, $\mathcal{Y}_{12}$ sent from B faithfully to A.
– Choose a (random) $\widetilde{x} \neq x_0$ so that that $H(\widetilde{x}) \neq X_0$.
– Send $\widetilde{x}$, X_1, $\mathcal{X}_{12}$ to B, i. e., replace the value x_0 sent by A with $\widetilde{x}$.

Since $H(\widetilde{x}) \neq X_0$, B initiates resynchronization after A has already updated its internal state to $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$, and we are in similar situation as discussed in Section 3.2.

4.2 First resynchronization (unsuccessful)

In this second step of the attack, F extracts the value x_1 from A, using the resynchronization process as shown in Figure 7.

A's internal state: $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$
A	***F***	***B***
Choose a random x_3, and form $X_3 := H(x_3)$, $\mathcal{X}_{23} := H(x_2, X_3)$.		Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$.
Receive y_0, Y_1, $\mathcal{Y}_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Send y_0, Y_1, $\mathcal{Y}_{12}$.
Send x_1, X_2, $\mathcal{X}_{23}$	$\xrightarrow{\widetilde{x}, X_1, \mathcal{X}_{12}}$	Since $x^*_{-1} \neq \widetilde{x}$, and $H(\widetilde{x}) \neq X_0$, initiate resynchronization.

Fig. 7. Second step of the attack: unsuccessful resynchronization

Thus F's actions in this step of the attack can be summarized as follows:

– Forward the values y_0, Y_1, $\mathcal{Y}_{12}$ sent by B faithfully to A.
– Receive x_1, X_2, $\mathcal{X}_{23}$ from A.
– Send $\widetilde{x}$, X_1, $\mathcal{X}_{12}$ to B, i. e., the same values as above.

Since y_0 and Y_1 match what A has stored, A refreshes the value $\mathcal{Y}_{12}$ with the new one sent by B. Recall that B has two sets of conditions to check as shown in Figure 3. As B has not accepted a password from A yet, we clearly have $x_{-1} \neq \widetilde{x}$ and the first condition is not met. Further, we have $H(\widetilde{x}) \neq X_0$, so the second condition is not met either. Hence the resynchronization is unsuccessful and B initiates resynchronization a second time. Note that at this point, F knows both x_0 and x_1.

4.3 Second resynchronization (successful)

During this second resynchronization, B will update its internal state. In preparation of the subsequent forgery, F binds the x_1-value received from A to F's own value $\widetilde{x}$. Figure 8 delineates the sequence of events during this second (successful) resynchronization.

A's internal state: $(x_1, x_2, X_1, X_2, \mathcal{X}_{12}, y_0, Y_1, \mathcal{Y}_{12})$		B's internal state: $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, \perp, X_0, \mathcal{X}_{01})$
A	***F***	***B***
Receive y_0, Y_1, $\mathcal{Y}_{12}$.	$\xleftarrow{y_0, Y_1, \mathcal{Y}_{12}}$	Choose a random y_2 and form $Y_2 := H(y_2)$, $\mathcal{Y}_{12} := H(y_1, Y_2)$. Send y_0, Y_1, $\mathcal{Y}_{12}$.
Choose a random x_3 and form $X_3 := H(x_3)$, $\mathcal{X}_{23} := H(x_2, X_3)$.	$\xrightarrow{x_0, X_1, \widetilde{\mathcal{X}}}$	Verifies that $H(x_0) = X_0$, $H(x_0, X_1) = \mathcal{X}_{01}$, then updates internal state.

Fig. 8. Third step of the attack: successful resynchronization

Summarizing, F does the following:

- Forward the values y_0, Y_1, $\mathcal{Y}_{12}$ sent by B faithfully to A.
- For the random $\widetilde{x}$ from the first step of the attack, form $\widetilde{X} := H(\widetilde{x})$ and $\widetilde{\mathcal{X}} := H(x_1, \widetilde{X})$.
- Send x_0, X_1, $\widetilde{\mathcal{X}}$ to B.

Since y_0 and Y_1 match what A has stored, A refreshes the value $\mathcal{Y}_{12}$ with the new one sent by B once again. As $H(x_0) = X_0$ and $H(x_0, X_1) = \mathcal{X}_{01}$, the second set of B's conditions is met, and B updates its internal state to $(y_0, y_1, Y_0, Y_1, \mathcal{Y}_{01}, x_0, X_1, \widetilde{\mathcal{X}})$. Hence, the second resynchronization is successful, and F can initiate an execution of the message recognition protocol with B.

4.4 Executing the message recognition protocol a second time

In the final step of the attack, F uses x_1 and the committing hash value $\widetilde{X}$ with the $\widetilde{x}$ chosen earlier. As seen in Figure 9, only F is communicating with A at this stage, and the message $\tilde{m}$ can chosen arbitrarily (with $m \neq \tilde{m}$ to achieve indeed a forgery).

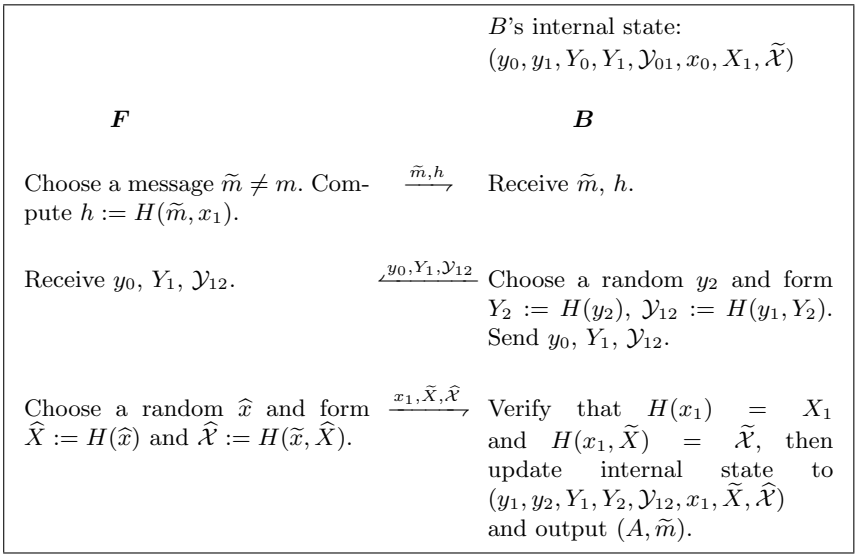

Fig. 9. Fourth step of the attack: inserting a forged message

The actions of F in this last part of the attack can be summarized as follows:

- Choose a message $\widetilde{m} \neq m$ and compute $h := H(\widetilde{m}, x_1)$.
- Send $\widetilde{m}$, h to B.
- Receive y_0, Y_1, $\mathcal{Y}_{12}$ from B.
- Choose a random $\widehat{x}$ and form $\widehat{X} := H(\widehat{x})$ and $\widehat{\mathcal{X}} := H(\widetilde{x}, \widehat{X})$.
- Send x_1, $\widetilde{X}$, $\widehat{\mathcal{X}}$ to B.

5 Conclusion

The above discussion shows that the message recognition protocol suggested by Mashatan and Stinson in [MS08] does not provide the intended security guarantees: the resynchronization procedure can be abused to provoke a situation where the protocol does not recover and enables a successful forgery attack. Consequently, the protocol from [MS08] should not be used in the present form.

Acknowledgments

We would like to thank the Fields Institute in Toronto and the organizing committee of the Fields Cryptography Retrospecive Meeting in 2009: the

financial support for attending this meeting enabled us to meet Atefeh Mashatan who made us aware of the work in [MS08]. We are indebted to Atefeh for helpful discussions.

References

[ABC+98] Ross Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, and Roger Needham. A New Family of Authentication Protocols. *Operating Systems Review*, 32(4):9–20, 1998.

[LZWW08] Stefan Lucks, Erik Zenner, André Weimerskirch, and Dirk Westhoff. Concrete Security for Entity Recognition: The Jane Doe Protocol. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology – INDOCRYPT 2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 158–171. Springer-Verlag, 2008.

[Mit03] Chris J. Mitchell. Remote User Authentication Using Public Information. In Kenneth G. Paterson, editor, *Cryptography and Coding, 9th IMA International Conference*, volume 2398 of *Lecture Notes in Computer Science*, pages 360–369. Springer-Verlag, 2003.

[MS08] Atefeh Mashatan and Douglas R. Stinson. A New Message Recognition Protocol for Ad Hoc Pervasive Networks. In Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong, editors, *Cryptology and Network Security, 7th International Conference, CANS 2008*, volume 5339 of *Lecture Notes in Computer Science*, pages 378–394. Springer-Verlag, 2008.

[SA00] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, *Security Protocols, 7th International Workshop*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–182. Springer-Verlag, 2000.

[WW04] André Weimerskirch and Dirk Westhoff. Zero Common-Knowledge Authentication for Pervasive Networks. In Mitsuru Matsui and Robert Zuccherato, editors, *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 73–87. Springer-Verlag, 2004.