RECOGNITION IN AD HOC PERVASIVE NETWORKS

ATEFEH MASHATAN[1] AND DOUGLAS R. STINSON[2]

Abstract. We examine the problem of message and entity recognition in the context of ad hoc networks. We review the definitions and the security model described in the literature and examine previous recognition protocols described in [1], [2], [3], [7], and [8]. We prove that there is a one to one correspondence between non-interactive message recognition protocols and digital signature schemes. Hence, we concentrate on designing interactive recognition protocols.

We look at [3] in more detail and suggest a variant to overcome a certain shortcoming. In particular, in case of communication failure or adversarial disruption, this protocol is not equipped with a practical resynchronization process and can fail to resume. We propose a variant of this protocol which is equipped with a resynchronization technique that allows users to resynchronize whenever they wish or when they suspect an intrusion.

1. Introduction

Entity recognition is a security notion weaker 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. It should also provide uniqueness, that is the corroborative evidence obtained in this process should uniquely determine 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, on the other hand, provides data integrity with respect to the data origin and 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. There are proposals for entity recognition and message recognition protocols in the literature, using both public-key and secret-key cryptography.

Public-key techniques such as digital signature schemes solve the problem of recognition easily. However, using these techniques in some scenarios, such as ad hoc pervasive networks, may be very costly. For instance, in an ad hoc network, there may be no pre-deployed authentic information available. 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

Date: May 27, 2008.

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

.

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 realistic.

RFID tags and sensor network motes in a hostile environment are examples of cases where public key techniques and secret key transmissions are not practical due to aforementioned reasons. Recognition protocols have applications in secure routing methods. In such scenarios, it may be unnecessary to provide the stronger security guarantees of entity and message authentication under reasonable assumptions. In fact, recognition is often all that is required in a dynamic environment. Hence, the weaker security of entity and message recognition is often pursued. Moreover, there are claims in the literature, [8], suggesting that in such an environment (where no pre-established authentic information exists and without the presence of a trusted third party), achieving anything more than recognition security is not possible.

We assume that two channels are accessible for communication: an insecure broadband channel and an authenticated narrow-band channel. We adapt the notation of [6] and [5] for illustrating these channels: the insecure channel and the authenticated channel are denoted by $\rightarrow$ and $\Rightarrow$, respectively. The latter is sometimes referred to as the manual channel. We use the narrow-band channel for the initial session between two users. Later sessions occur over the insecure channel.

The adversary has full control over the broadband channel. That is, the adversary can listen to any messages sent over the broadband channel, modify the messages sent via this channel, stall the message from being delivered, and inject a new message into this channel at any time. However, the adversary remains passive on the narrow-band channel.

We have two small devices, Alice and Bob, in a hostile environment. They have previously met in a secure environment. Now, Alice wants Bob to recognize her or recognize the messages sent from her to Bob. The adversary Eve is trying to make Bob recognize Eve as Alice, or recognize messages from Eve as sent from Alice, where Alice has never sent those messages. We further assume that Eve can make Alice send messages of Eve's choice.

The rest of the paper is organized as follows. Section 2 is devoted to examining previous recognition protocols in [1], [2], [3], [7], and [8]. In particular, some shortcomings of these recognition protocols in the context of ad hoc networks are described.

In Section 3, we look at the protocol proposed in [3] in more detail and suggest some improvements. Moreover, in case of communication failure or adversarial disruption, none of these protocols are equipped with a practical resynchronization process. We propose a variant of the recognition protocol in [3] which includes resynchronization technique.

Further, we prove that there is a one to one correspondence between non-interactive message recognition protocols and digital signature schemes in Section 4.

2. Previous Recognition Protocols

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 ad hoc pervasive networks where devices have low computation power and low communication bandwidth.

The Guy Fawkes protocol was proposed by Anderson et. al. in [1]. There are two variants of this protocol suggested and a one-way hash function is deployed in both variants. In the first variant of the Guy Fawkes protocol, 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 these 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 of the Guy Fawkes protocol 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 in ad hoc networks, 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 'Remote User Authentication Protocol', was introduced by Mitchell in [7]. 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 in 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 computations and communication here is large compared to other protocols that are providing entity or message recognition and it may not be suitable for ad hoc network settings with low power devices.

Weimerskirch et. al. introduced a protocol called Zero Common-Knowledge (ZCK) protocol in [8]. 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. implemented the ZCK protocol and published the results in [2]. The provided measurements and observations from this implementation provided a proof-of-concept. Low computational power, low code space, low communication bandwidth, low energy resources, are among the main requirements of a recognition protocol designed for an ad hoc pervasive network setting. The measurements resulted from this implementation proved that the ZCK protocol exhibits the aforementioned requirements. They note that denial-of-service and memory complexity are areas of concern and need to be addressed or improved upon in the future.

Note that [2] investigates the practicality of the ZCK protocol and does not investigate its security properties. That is, it relies on the security proof presented in [8]. However, in [3], Lucks

et. al. found a flaw in the security proof of this protocol and presented an attack against the ZCK protocol. Furthermore, they proposed a modification to fix the flaw.

2.1. **LZWW05 Protocol.**

As noted above, in [3], Lucks et. al. found an attack against the ZCK protocol of [8] and pointed out the flaw in the security proof of this protocol. Further, using the same idea of using values in a hash chain as keys for MACs, they proposed a message recognition protocol which is a modification of the original ZCK protocol.

They consider a cryptographic hash function $h : \{0,1\}^s \to \{0,1\}^s$, as a one-way hash function, and a message authentication code $\text{MAC} : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^c$. Typical values are suggested to be $s \geq 80$ and $c \geq 30$. Further, n is fixed to be the maximum number of messages to be authenticated. In other words, the maximum number of sessions is fixed to be n. Alice and Bob randomly choose a_0 and b_0, respectively. Then, they respectively form $a_i = h(a_{i-1})$ and $b_i = h(b_{i-1})$, $i = 1, \ldots, n$.

In the initialization phase, Eve is assumed to be passive. Hence, we can denote this channel, in accordance with our notation, by $\Rightarrow$. Alice and Bob will exchange a_n and b_n over the authenticated channel during this phase.

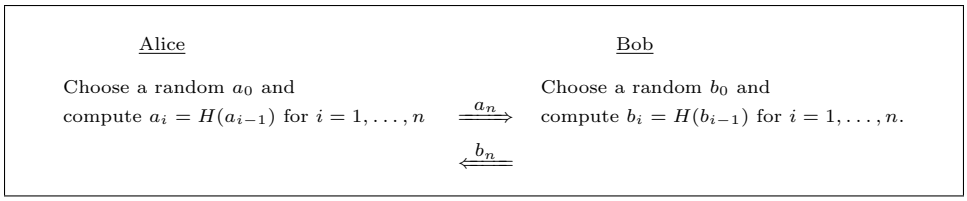

FIGURE 1. Initialization Phase of LZWW05 Recognition Protocol

After the initialization phase, there are n sessions denoted by $n-1, \ldots, 0$, starting from session $n-1$ and moving down to lower values one at a time. In session i, Alice authenticates the message m_i using a_i as the key for the MAC. Once Bob authenticates himself to Alice by revealing b_i, Alice reveals a_i and allows Bob to verify and accept this new key and the authenticity of the message m_i. When a key k is accepted, it is denoted by accept-key(k). Moreover, commit-message(m, i) indicates that Alice commits to a message m in session i, and accept-message(m, i) indicates that Bob accepts m as authentic and fresh in session i. After a successful session of the protocol, Alice and Bob will "move down" in the hash chain, using a_{i-1} and b_{i-1} for session $i-1$.

Formally, Alice and Bob keep the following information as their internal states:

- i, the session counter,
- b_{i+1}, the most recently accepted value of Bob's hash chain kept by Alice, and a_{i+1}, the most recently accepted value of Alice's hash chain kept by Bob (hence accept-key(b_{i+1}) and accept-key(a_{i+1}) have occurred already).
- a one-bit flag, to determine the program states $A0$ and $A1$, or $B0$ and $B1$ for Alice and Bob, respectively.

Session i of the protocol is executed as follows:

$A0$ (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 $b_i := b'$; accept-key(b_i); send a_i; set $i := i - 1$; goto $A0$,
else goto $A1$.

$B0$ (Bob's initial program state) Wait for a message (d_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
- $a_i := a'$; 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);
- set $i := i - 1$; goto $B0$

else goto $B1$.

Figure 2 depicts the LZWW05 protocol. We analyze this protocol in more detail in Section 3 and point out its shortcomings in case of adversarial disruption or communication failure. Further, we propose a new variant of the recognition protocol of [3] which incorporates a resynchronization technique allowing a full recoverability of the protocol.

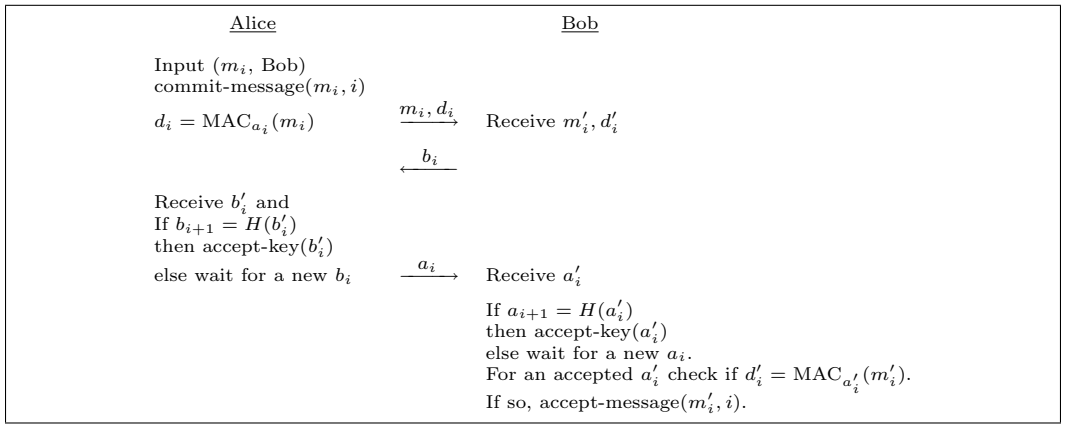

FIGURE 2. LZWW05 Entity and Message Recognition Protocol

The extended abstract, [3], does not contain the security proof of this protocol and the reader is referred to the full version of the paper [4] for the security proof.

3. An Improvement to the LZWW05 Protocol

In [3], 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, [3] assumes 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.

In particular, an adversary can easily manipulate one user to move forward to the next time-frame, while the other user is still in the previous time-frame. In this case, Alice and Bob will not be able to successfully execute the protocol anymore. For instance, Bob could get trapped in a case where the condition in the program state $B1$ fail and he stays in $B1$ in a vicious circle. The way the protocol is stated suggests that Bob could stay in program state $B1$ forever! This situation is depicted in Figure 3. As illustrated in this figure, upon reception of the new message m'_i and d'_i, Bob will automatically move forward to the next time-frame whereas Alice is still in the previous time-frame. Furthermore, since $a_{i+1} \neq H(a'_i)$, Bob is instructed to wait for a new a_i. Moreover, 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, even if it is really sent by Alice! Therefore, he is "trapped" in state $B1$. It is suggested in [3] that, having waited for too long to receive the correct a_i, Bob sends b_i again. However, this is not going to overcome the problem here. Alice has not initiated the session and is not anticipating b_i.

Eve		Bob
Choose random m'_i and d'_i.	$\xrightarrow{m'_i, d'_i}$	Move to the next time-frame upon reception of the
	$\xleftarrow{b_i}$	new message and send 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 3. Eve "trapping" Bob in state $B1$

Similarly, Alice could get stuck in program state $A1$ for an indeterminate period of time. This situation is depicted in Figure 4. Note that the protocol has failed in both cases while the adversary does not need to continue changing the information. Indeed, the adversary can honestly relay the rest of the messages, yet Alice and Bob will no longer be able to successfully execute the protocol. This contradicts the recoverability notion that the paper is claiming.

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

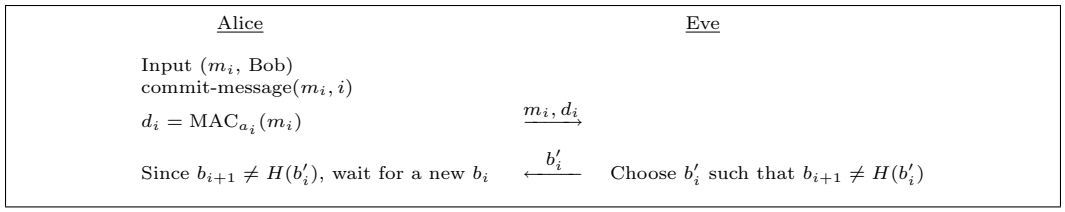

FIGURE 4. Eve "trapping" Alice in state $A1$

the initialization phase. For instance, the sensor motes 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 motes maybe 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 in [3] and write H^j, $j \geq 1$, to denote the case when the hash function H is applied j times iteratively.

3.1. **Description of our Protocol.**

The internal state of Alice and Bob includes:

- i_A and i_B, counters pointing 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 $d_i = \mathrm{MAC}_{a_i}(i \| m_i)$;
- 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.

- 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 $\mathrm{MAC}_{a_{i'}}(i' \| m'_{i'}) = d_{i'}$
    then accept $m'_{i'}$ as authentic in session i';
  else initiate the resynchronization process.

Figure 5 illustrates this protocol. Let us first highlight the differences between this protocol and the protocol of [3]. In the internal states of Alice and Bob, the session counter i is replaced by i_A and i_B to incorporate adversarial ability to manipulate a party to increment 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. Moreover, 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 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 [3], the session counter is not being transmitted or committed to by either party. However, we require that Alice commits to i_A and transmit 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 the 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 $i_{acceptA} = i_{acceptA} = i + 1$. 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 increment 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 $j_B \neq 1$.

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 LZWW05 protocol, found in [4], 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.

3.2. **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

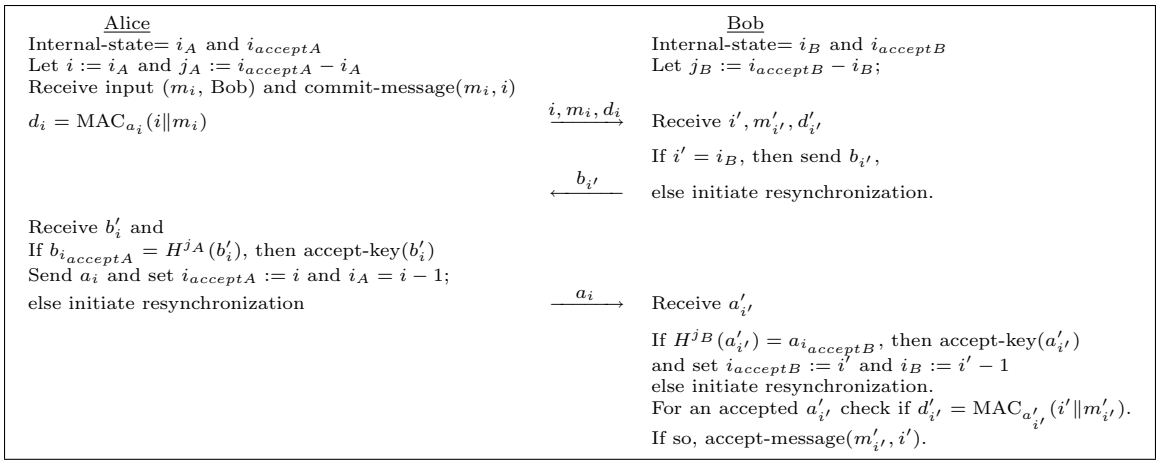

FIGURE 5. Our Proposed Variant of LZWW05 Protocol

computes $I_A := \min\{i :$ Alice has revealed $a_i\} - 1$ and, similarly, Bob computes $I_B := \min\{i :$ Bob has revealed $b_i\} - 1$. 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. Alice will let $i_A := \min(I_A, I'_B)$ and Bob will similarly find $i_B := \min(I'_A, I_B)$. Figure 6 depicts the resynchronization process.

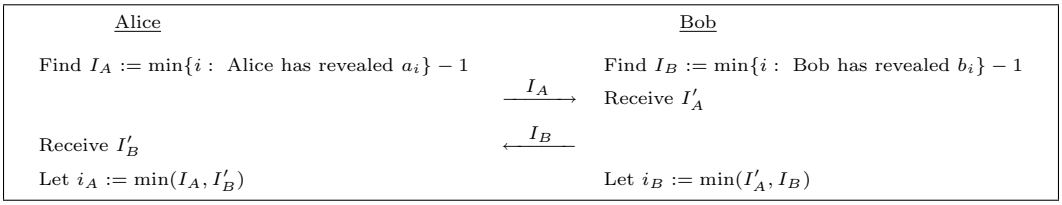

FIGURE 6. Resynchronization Process for the Proposed Protocol

Note that an active adversary can always disrupt the synchronizations. 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 the synchronization is achieved. On the other hand, we will show that intrusions of an active adversary during the resynchronization stage, is going to be detected by either Alice or Bob. Consider the first execution of the protocol immediately after the resynchronization, depicted in Figure 7. We now show that the case of $i_A \neq i_B$ will be detected by one of Alice or Bob, unless the adversary has found unrevealed preimages of particular values in the hash chain.

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 set 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 preimages $a_{i_A}, a_{i_A-1}, \ldots, a_{i_B+1}$. Hence, as long as finding pre-images of H is a hard task, the adversary will be detected with high probability. As a result, we obtain the following theorem.

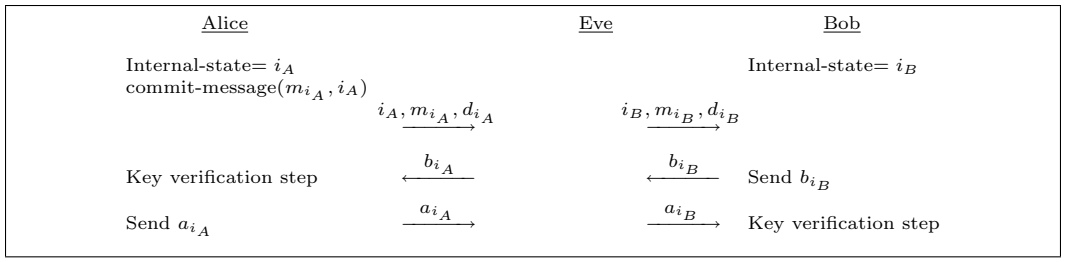

FIGURE 7. The first execution after the resynchronization

Theorem 1. *Let H be a pre-image resistant hash function in the protocol of Figure 5. Consider a polynomially bounded adversary who changes the values of I_A or I_B in the resynchronization process of Figure 6, resulting in $i_A \neq i_B$. A successful such intrusion can only occur with a negligible probability.*

4. Non-interactive Message Recognition Protocols and Digital Signature Schemes

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

4.1. **Non-interactive Message Recognition Protocols.**

As was mentioned before, a message recognition protocol provides data integrity with respect to the data origin and assures the recipient that the same entity who was initially the source of a recognized message is sending messages in future sessions. The initialization phase occurs over an authenticated channel. In order to authenticate future messages, parties use the insecure channel in the future sessions. A general non-interactive message recognition protocol, where all flows are going from Alice to Bob, consists of two flows. The first flow refers to 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 its commitment (and possibly some other information) are all being transmitted in one flow. Hence, one should not reveal keys, such as a_i in the hash chain, in these protocols. Otherwise, the adversary having seen the revealed key will stop this single flow and commit to a message of her own using this key. This implies that there is no point in using hash chains or any form of chains in the non-interactive setting since the chains can only be useful when you actually reveal them.

Figure 8 depicts a general non-interactive message recognition protocol. Here, f needs to be a one-way function to make the impersonation impossible. That is, it should be hard to find a given the value of A.

The protocol is described in terms of two functions, denoted by *compose* and *decompose*. The function *compose* can be a randomized algorithm. Note that any non-interactive protocol can be put in this form. It is required that $decompose(c', A) = \perp$ with high probability if $c' \neq compose(M, a)$ for some message M and $A = f(a)$. Moreover, it is required that $decompose(c, A) = M$ when $c = compose(M, a)$.

4.2. **Signature Schemes.**

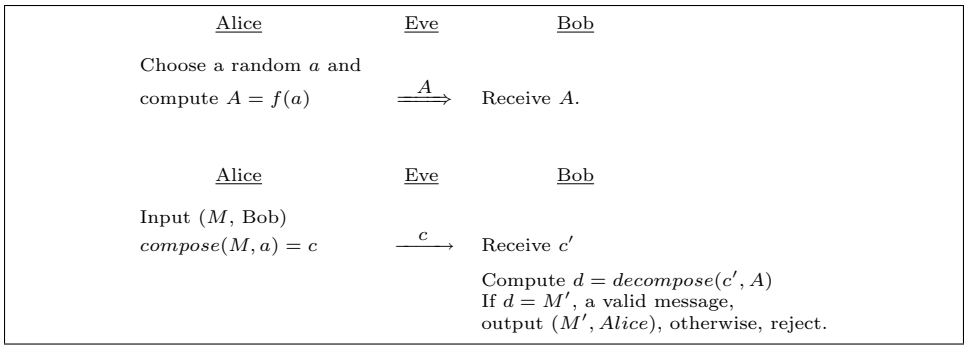

FIGURE 8. A Non-interactive Message Recognition

Typically, there are three algorithms in a signature scheme: key generation, sign and verify. They work as follows:

- 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.
- On input message m and a secret key SK, the signing algorithm $\mathcal{S}$ outputs a signature s. ($\mathcal{S}$ may be a randomized algorithm.)
- On input a message m, a public key PK, and a signature s, the signature verifying algorithm, $\mathcal{V}$, either outputs 1, indicating that s is a valid signature for m given PK, or it rejects s by outputting 0.

These steps are illustrated in Figure 9.

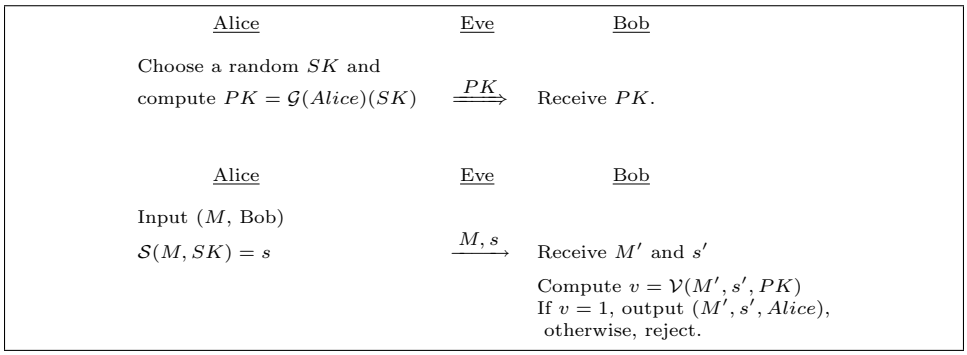

FIGURE 9. A Digital Signature Scheme

It is required that the following properties hold for these algorithms:

- Correctness: A signature s of m that is computed honestly using the secret key SK should be accepted by the verifying algorithm using the associated public key PK. In other words, for all m, PK, and SK
$$\mathcal{V}(m, PK, \mathcal{S}(m, SK)) = 1.$$
- It should be difficult for any polynomially bounded adversary, to forge valid signature(s) knowing only the public key PK, and the three algorithms.

4.3. **Equivalence of Non-interactive Message Recognition Protocols and Digital Signature Schemes.**

We prove the equivalence of digital signature schemes and non-interactive message recognition protocols in the following theorem.

Theorem 2. *Given functions f, compose, and decompose, any non-interactive message recognition protocol can be transformed to a digital signature scheme. Conversely, any digital signature scheme, with functions $\mathcal{G}, \mathcal{S}$, and $\mathcal{V}$, can be transformed to a non-interactive message recognition protocol.*

Note that it was previously known that a signature scheme could be used to construct a non-interactive recognition scheme. The converse result is new.

Proof. We first show that a non-interactive message recognition protocol gives rise to a signature scheme. We let $SK := a$ and $\mathcal{G} := f$, which implies that $PK = A$. Further, we compute the $\mathcal{V}$ and $\mathcal{S}$ functions as follows:

- $\mathcal{S}(M, a) = \mathit{compose}(M, a)$, where $a = SK$, and
- $\mathcal{V}(M, A, s) = \mathit{decompose}(c, A)$, where $c = M \| s$ and $A = PK$.

It is now easy to see that the desired properties of the *compose* and *decompose* functions of the non-interactive message recognition protocol translate to the correctness and soundness properties of the signature scheme.

Similarly, we show that a signature scheme can be used as a non-interactive message recognition protocol by the following assignments:

Let $a := SK$ and $f := \mathcal{G}$, which implies that $A = PK$. Let the *compose* and *decompose* functions be defined as follows:

- $\mathit{compose}(M, a) = M \| \mathcal{S}(M, a)$, where $a = SK$, and
- $\mathit{decompose}(c, A) = \mathcal{V}(M, A, s)$, where $c = M \| s$ and $A = PK$.

□

5. Conclusions

The problem of message and entity recognition for ad hoc network scenarios is investigated in this paper by first reviewing the definitions and the security model described in the literature. Previous recognition protocols were revisited and their shortcoming in such scenarios were pointed out. The most recent work in this setting, and the best one in terms of practicality, is presented in [3].

We looked at this protocol in more detail and described a situation where the protocol fails to recover after the adversary's intrusion. We further suggest a variant of this protocol to overcome this problem. In particular, in case of communication failure or adversarial disruption, this protocol is not equipped with a practical resynchronization process and can fail to resume. Our proposed variant is equipped with a resynchronization technique that allows users to resynchronize whenever they wish or when they suspect an intrusion. Hence, our proposed protocol can fully recover.

Finally, we prove that any particular design of a non-interactive recognition protocols can be transformed into a digital signature scheme, and vise versa. Digital signature schemes involve

computations that are expensive for a low-power environment. Hence, this suggests that attention should be drawn to designing interactive recognition protocols for ad hoc pervasive networks.

Acknowledgments

Douglas R. Stinson's research is supported by NSERC discovery grant 203114-06.

References

[1] Ross Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, and Roger Needham. A new family of authentication protocols. In *ACMOSR: ACM Operating Systems Review*, volume 32, pages 9–20, 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. Submitted for publication.

[5] Atefeh Mashatan and Douglas R. Stinson. Interactive two-channel message authentication based on interactive-collision resistant hash functions. Technical Report 02, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2007.

[6] Atefeh Mashatan and Douglas R. Stinson. Noninteractive two-channel message authentication based on hybrid-collision resistant hash functions. *IET Information Security*, 1(3):111–118, September 2007.

[7] 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.

[8] 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.