A new generic protocol for authentication and key agreement in lightweight systems

Naïm Qachri[1], Frédéric Lafitte[2], and Olivier Markowitch[1]

[1] Département d'Informatique, Université Libre de Bruxelles, CP212, boulevard du Triomphe, 1050 Brussels, Belgium
nqachri@ulb.ac.be,olivier.markowitch@ulb.ac.be
[2] Royal Military Academy, Department of Mathematics Renaissancelaan 30, 1000 Brussels, Belgium
frederic.lafitte@rma.ac.be

Abstract. In this paper, we propose a new generic authenticated key agreement protocol where the master secret is automatically renewed based on a sequence of hash values, thus providing the system with an extended cryptoperiod. The focus of this work is to formally assess the security offered by the protocol's key renewing in the case of a long term use of the system. The formal analysis is carried using the automated tools ProVerif and AVISPA. The protocol is designed to be implemented on devices with limited computing and storage resources.

Key words: Mutual authentication, Key agreement protocol, Wireless communication security, Cryptoperiod

1 Introduction

Since a decade, we have seen many developments to strengthen the security of wireless embedded communication systems: these improvements intended to correct problems related, for example, to defective cryptographic primitives or protocols [17, 7, 10] (e.g. the 4-way handshake protocol defined for the Wimedia MAC layer standard [16] used to authenticate and generate session keys). Nowadays, the last versions of the Wifi and Wimax standards include the use of EAP [1] declined in different versions (LEAP – EAP using a Radius Server –, EAP-TTLS, etc.). In practice, EAP is not well suited for constrained environments such as handheld devices, short-range communication systems or even domestic wireless LAN devices. The reason being that many versions of EAP use certificates, public key encryption or exhaustive exchanges of information, that are not always appropriate for lightweight wireless devices. For example, certificate revocation is not viable in point to point communications.

Moreover, wireless communication protocols are dedicated to the specific technology considered. This lack of genericity makes the security evaluation of such protocols more cumbersome and also less re-usable. Furthermore, those protocols do not address the issue of the master secret cryptoperiod. This weakness has already been used in order to mount practical attacks in [17, 10]: instead of

renewing the key used by RC4, the WEP protocol uses a different IV (Initalization Vector). However, the number of IVs is not large enough to prevent key re-use, which is a serious weakness for stream ciphers.

Furthermore, the existing systems proposed in the literature do not meet all of the needed requirements; indeed, the existing solutions either do not combine authentication and key generation [14, 5], use public-key cryptography [20, 21, 12, 9], use a trusted entity [14, 15], or combine artificially an authentication scheme [13] with a key transport protocol [14] (involving more transfers without gaining more security). Similar authentication mechanisms used for [13] are used in [3, 4]. The ISO/IEC 9798-2 and ISO/IEC 11770-2 protocols (ISO standards that define some key agreement protocols) differ from our protocol since they do not authenticate the exchanged messages and they consider only single execution of the protocols. Therefore, because of the lack of messages authentications, the information exchanged during these protocols (that are used to produce the session keys) are not protected against modifications, and may be exposed to denial of services attacks. Moreover, most of the protocols does not include a session key confirmation procedure as well as a synchronization mechanism (i.e. in case of repetitive establishments of sessions).

Contribution. According to the paragraphs above, we are after a key agreement protocol that relies only on symmetric key cryptography and satisfies all the requirements for contributive authenticated key exchange in wireless communication systems. In particular, we focus on the protocol lifecycle by introducing an automated key renewing mechanism. The security of our proposal is analyzed in a Dolev-Yao model [6] where the adversary has corruption capabilities. We use the tools AVISPA and ProVerif in order to automate the analysis. Our protocol is generic since we do not instantiate the underlying cryptographic primitives. It is considered lightweight since no asymmetric algorithms are used. Therefore, in this symmetric key setting, our protocol best fits point to point communications, or centralized networks, where a device interacts with a limitted number of participants.

Outline. The paper is divided in six sections. The next section exposes the assumptions, definitions, and notations that will be used along the paper. The third section presents a general description of the protocol and a formal definition of the protocol and the subprotocols is given. The fourth section presents a formal analysis of the security of the protocol. The fifth section describes an interesting security finding. In the sixth section, we summarize some future works for a larger exploitation of our protocol.

2 Preliminaries

In this section we define the security requirements for authenticated key exchange protocols as well as the adversarial capabilities. We also introduce general assumptions and notations.

2.1 Requirements

Common security requirements for key agreement include:

Contributiveness: the key exchange is said to be contributive if Alice and Bob contributed equally to the computation of the new session keys (and the renewing of the long-term key).

Backward security: future sessions remain secure even if secrets used in past sessions are corrupted. Different types of secrets can be leaked in a session. The type of secret that is leaked is used to caracterise the type of corruption (see types of corruption below).

Forward security: past sessions remain secure even if future sessions are corrupted. Again, different kinds of corruptions can be considered.

Mutual authentication: the mutual authentication is a mechanism that allows to authenticate two devices to each other. In our case, this property is obtained through a *challenge-response* mechanism.

Key confirmation: the purpose of key confirmation is to make sure that both Alice and Bob derived the same session keys. Many existing protocols do not satisfy this property. Obviously, this verification procedure must not reveal the underlying generated key and can be achived through a *challenge-response* mechanism.

In addition to the rather common properties defined above, our protocol has the following property.

Key renewing: another desirable property (see intro) consists in the automated renewing of the master secret. We consider this key renewing procedure secure as long as the new key remains secret.

The requirements defined above only make sense when considered together with a specific adversary. In our case, the (active) adversary has Dolev-Yao capabilities [6] over all communication channels, i.e., it is capable to read, delay, delete, insert messages from/to the channels used in the execution of the protocol.

We also assume that the adversary is able to corrupt participants, i.e., to learn their secret values. Secrets involved in key exchange protocols are often categorized as short term (ephemeral) secrets and long term (master) secrets. In our case, the ephemeral secrets correspond to the nonces and the master secret consists of the chain of hash values. This leads to the consideration of three kinds of corruptions.

Types of corruptions: Corruptions that leak an ephemeral secret are referred to as type I corruptions whereas those leaking the master secret are referred to as type II corruptions. Type III corruptions correspond to the case where both ephemeral and master secrets are leaked.

In section 4, we assess to what extent each type of corruption can be used to compromise the requirements defined above, in particular, forward and backward secrecy. This analysis is conducted in a symbolic framework.

2.2 Assumptions

We suppose that Alice and Bob (wireless devices that know each other) share a symmetric key that has been secretly exchanged during an association step that happens before any exchange that would occur between the two devices. It is assumed that this association step is secure and therefore does not leak any information about the shared secret. Furthermore, the association step is not considered in the protocol because this step is strongly dependent of the technology considered and will remove a part of the genericity of the protocol. Moreover, it is assumed that the secret, shared by Alice and Bob, is not already shared by them with any other devices.

It is also assumed that the devices are tamper resistant (i.e. an attacker cannot physically read or modify the secrets stored in the devices).

2.3 Notations

The notations used in our protocol are the following:

- h_i denotes that the hash function H is applied successively, i times, according to the following construction:

$$h_1 = H(s)$$
$$h_i = H(h_{i-1} \parallel s) \qquad \forall i > 1$$

- $E_k(m)$ denotes a symmetric bloc encryption of the message m with the secret key k;
- $MAC_k(m)$ denotes the result of a keyed hash function applied on a message m;
- s denotes the secret shared between Alice and Bob during the association step;
- r_A and r_B denotes the random nonces chosen and sent respectively by Alice and Bob during a session of the key agreement protocol;
- $LSB_i(m)$ is a function that truncates m to its i *least significant bits*.

3 The protocol

3.1 General description of the protocol

Based on the initial secret (exchanged at the association step), keys for authentication and encryption are generated and shared between Alice and Bob.

The protocol is designed in order to avoid that an attacker, who discovers the secrets of a session of the protocol, can deduce the secrets that will be computed during the following sessions. The secret values of the different sessions are computed on the basis of a chain of hash values. During an initialization step (that takes place after the association step), Alice and Bob realize the computation of n consecutive hashing on the initial shared secret. Those hashed

values will be used to authenticate Alice and Bob and to generate the session keys as described hereafter. After the initialization step, when a session must be set, Alice invokes a 3-step main subprotocol that ensures mutual authentication by the means of challenge-response techniques, session keys generated between Alice and Bob and desynchronization resistance.

Since the produced secret hashed values are in a limited number, when only three hash values remain, a new secret is computed using those values and this new secret key is used to create a new chain of hash values that will be used for the next sessions of the protocol. The chain of hash values implies a notion of lifecycle of secret keys.

The key renewing procedure and the generation of keys during the key agreement are based on a secure key generation mechanism similar to the HMAC-Key Derivation Function (`HDKF` [11]).

3.2 The protocol life cycle

The protocol begins with the two following steps:

- the association step (where a secret s is exchanged);
- the initialization step (where some or all of the n hashed values, from the secret s, are computed in order to speed up further hash computations in the protocol and i is initialized to 1 by Alice and Bob).

$$h_1, \ldots, h_n \quad \text{and} \quad CC \leftarrow 1$$

Then, the life cycle of the protocol is described as follows :

- $\lfloor \frac{n}{3} - 1 \rfloor$ successive executions of the main protocol
  and/or *resynchronization protocol* (if needed)
- renewing protocol (where the initialization step is made again after the renewing)
- successive executions of the main protocol (a new cycle is launched)...

The *Cycle Counter* (CC) is a variable that counts the number of chains of n hashed values completely used in the life cycle of the protocol since the initialization step (where CC is set to 1). Within a cycle, a session of the main protocol is characterized by a number i. The concatenation $CC \parallel i$ is a unique identifier of a session of the protocol between Alice and Bob.

The initialization step, made after the association step, consists in computing the chain of the n hashed values that will be used during the executions of the main subprotocol during a lifecycle. This initialization step is made again after a renewing of the secret.

3.3 The main subprotocol

When Alice and Bob have to initiate a new session i, being in the cycle CC, they execute the following main protocol:

$$\textbf{1}. \quad \textbf{Alice} \rightarrow \textbf{Bob} : ID_{Alice}, i, E_{h_{n-((i-1)\times 3)}}(m_1, H(m_1))$$
$$\text{where} \quad m_1 = (1, CC, i, r_A, ID_{Alice})$$

The message contains the first challenge under the form of a message to decrypt and verify. If the two devices share the same secret, then Bob can decrypt and verify it. The nonce, r_A, sent by Alice, has to be well chosen and contributes to the keys generation in a fair way in regards to Bob. The number i of the session is sent unencrypted to synchronize the two devices on the keys to use (see the resynchronization subprotocol). The hashing, at the end of the encrypted message, is computed to ensure that the message cannot be easily manipulated by the attacker regardless of the encryption algorithm.

Once Bob has decrypted the message, he chooses a second random nonce, r_B, and generates three keys by computing the following MAC:

$$(\mathbf{k_{SE}} \parallel \mathbf{k_{SA}} \parallel \mathbf{k_{conf}}) = LSB_q(MAC_{h_{n-((i-1)\times 3)-2}}(CC \parallel i \parallel r_A \parallel r_B \parallel h_{n-((i-1)\times 3)-1}))$$

where $\parallel$ is the concatenation operator. r_A and r_B are the contributions of respectively Alice and Bob in the computation of these three keys. q denotes the sum of the sizes of the three keys generated during a session of the protocol. $CC \parallel i$ is used to avoid replay attacks. $\mathbf{k_{SE}}$ is the key generated to encrypt the communication of the session that will take place between Alice and Bob and $\mathbf{k_{SA}}$ is the key generated to authenticate the packets transmitted during this communication.

The MAC algorithm has to be well dimensioned to generate enough bits for the three keys. On the basis of $\mathbf{k_{conf}}$, Bob creates a new challenge and sends it to Alice.

$$\textbf{2}. \quad \textbf{Bob} \rightarrow \textbf{Alice} : ID_{Bob}, i, E_{h_{n-((i-1)\times 3)}}(m_2, MAC_{k_{conf}}(m_2))$$
$$\text{where} \quad m_2 = (2, CC, i, r_B, r_A, ID_{Bob})$$

The challenge has the purpose to ensure that Alice can derive the good key and decrypt the message of Bob. From these keys, Alice can verify that she has derived the same keys than Bob if she is able to verify the MAC on the message. Alice can also authentify Bob, since only Bob knows the secret hashed value used to encrypt the message and authenticate. Furthermore, Bob sends the nonce r_A to prove that he has made the correct decryption of the first message.

$$\textbf{3}. \quad \textbf{Alice} \rightarrow \textbf{Bob} : ID_{Alice}, i, MAC_{k_{conf}}(3, ID_{Alice}, CC, i, r_A, r_B)$$

In this third message, Alice answers that she has well derived the keys and that the authentication of Bob succeeds, she provides also r_B to prove that she has made the correct decryption of the second message. At the end of the main protocol, Alice and Bob increment i.

3.4 The renewing subprotocol

Within a cycle, on the basis of n hash values, we can realize $\lfloor \frac{n}{3} - 1 \rfloor$ sessions of the main protocol. We use the last session of the protocol to generate a new secret value that will overwrite the previous shared secret between Alice and Bob (initially s).

The renewing is made when only three hash values (h_3, h_2 and h_1 remain). Alice and Bob run again the main subprotocol (see Figure 1) during which the new secret is computed from $h_1 = H(s)$ and s. This execution allows the exchange of r_A and r_B.

$$\mathbf{s_{new}} = MAC_{s_{old}}(CC \parallel r_A \parallel r_B \parallel H(s_{old}))$$

The shared secret hash values and the CC are computed for the future sessions of the next cycle of the main protocol:

$$h_1 = H(s_{new}), \ldots, h_j = H(h_{j-1}||s_{new}) \quad \forall j \in \{2, \ldots, n\}$$
$$\text{and} \qquad CC \leftarrow CC + 1; \qquad i \leftarrow 1$$

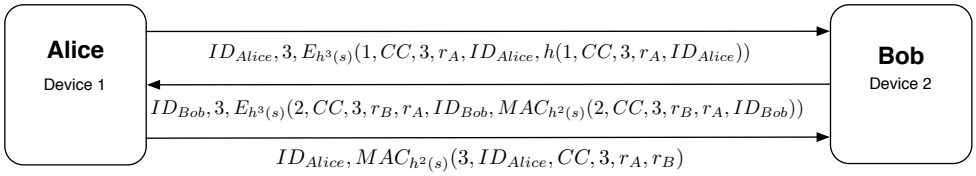

Fig. 1. The renewing protocol

3.5 The resynchronization protocol

If the two devices are desynchronized (i.e. if Alice and Bob consider a different value of i), they reveal their session values i and i'. In that case, the current session of the main protocol is aborted and a new session protocol is launched with a session value $max(i, i') + 1$.

We make the assumption that, in case of desynchronization, the devices cannot have different CC's, because it would mean that one of the two devices has done the renewing protocol without having incremented the variable CC (unless one of the devices were cloned).

4 Security Analysis

In this section, we use the automated protocol verification tools AVISPA and ProVerif in order to show that our protocol meets the requirements defined in section 2.1 when attacked by an active Dolev-Yao adversary with corruption capabilities.

The symbolic Dolev-Yao model abstracts away cryptographic operations on bits and considers only *symbols* in order to represent keys, nonces, messages, etc. Therefore, in this model, cryptographic primitives are considered ideal, since the adversary is able to recover the secret symbol (i.e. 100% of the secret bits) or not (i.e. 0% of the secret bits). In our case, since we do not instanciate the cryptographic algorithms used by our protocol, this abstraction is necessary for the analysis of the protocol.

AVISPA is used to establish the mutual authentication property whereas ProVerif is used to assess the impact of corruptions. In both cases, the three keys derived at the end of one session are considered as one unique symbol (being the concatenation of the three keys).

4.1 Analysis using AVISPA

AVISPA [19, 18] is an automated protocol verifier based on temporal logic. Its purpose is to check the security of protocols through a specification language named HLPSL. The language allows for the description of the protocol through roles and sessions of execution where the channel of communication meets the Dolev-Yao model [6]. We have chosen AVISPA, because it offers the opportunity to analyze our protocol in four *backends* (verification engines) and then to cover a large spectrum of possible attacks.

Since AVISPA does not define boolean or MAC functions, we have rather modeled the HMAC function (through its formal definition):

$$HMAC_k(\mathbf{m}) = H((k \oplus \mathbf{opad})||H((k \oplus \mathbf{ipad})||\mathbf{m}))$$

where $\oplus$ is the *exclusive-or* and $||$ the concatenation operation. The **opad** and **ipad** values are constant values, respectively composed of the repeated $0x5C$ and $0x36$ values, as long as the key k.

Because of the modeling of the HMAC with the use of *exclusive-or* operations, two of the four backends (i.e. SATMC and TA4SP) became not conclusive on the security of the protocol, because they cannot manage arithmetic or boolean operations used within the protocol specification. We have analyzed the protocol on parallel sessions where we have explicitly computed a chain of hashed values.

We have taken a particular care in defining the security goals within AVISPA and the complete formalization of the process of the mutual authentication. In the specification, we have authenticated both parties with the challenges provided by the exchange of their nonces r_A and r_B, but also on $\mathbf{k_{conf}}$ that confirms the good derivation of the key and the fact that they know the secret hashed values without revealing them.

The sessions were specified to consider various possible attacks such as *Man in the middle attacks*, or *replay attacks* (through parallel executions of sessions).

The protocol was also checked in order to find some way to attack the secrecy of the three keys generated or the mutual authentication of both parties. The tests have given safe (i.e. no attacks were found) results on two of the four

backends (the two other were non conclusive due to the use of boolean arithmetic). This validate that our protocol is secure, that both parties are mutually authenticated and that the lifecycle is secure (see the discussion section below). The complete specification of the protocol is given in append.

4.2 Analysis using ProVerif

ProVerif [2] is an automated cryptographic protocol verifier, mainly used to assess secrecy and authentication properties. The protocol may be specified in different formalisms, in particular an extension of the (typed) applied pi calculus. Next, the adversary's goals and capabilities are specified (see appendix B). Internally, ProVerif translates the protocol and the adversary's capabilities into first order logic formulas (i.e. Horn clauses). Finally, a dedicated resolution algorithm is used to output one of the three following outcomes:

– a (sound) confirmation that the adversary is unable to reach his goal
– the execution trace of a successful attack
– the inability of the tool to conclude

Fortunately, in many practical cases ProVerif is able to conclude in a few seconds. The main advantage of ProVerif lies in its ability to analyze an unbounded number of sessions.

We model the execution of three consecutive sessions in order to show whether leaking a secret during session i helps the adversary in the recovery of secrets from session $i-1$ (forward secrecy) or session $i+1$ (backward secrecy). At the end of each session, both participants use the exchanged key to encrypt a secret that is specific to the session. That is, the key established in session one is used to encrypt the symbol $\mathsf{secret1}$, the one established during session two encrypts the symbol $\mathsf{secret2}$, and so on. The adversary is then queried on those three secrets since recovering one of them is equivalent to recovering the corresponding session key. As mentionned in section 2.1, the types of secrets leaked are either ephemeral secrets (i.e. nonces), master secrets (i.e. values from the hash chain) or both master and ephemeral secrets, thus caracterising three kinds of corruptions, referred to respectively as "type I", "type II", and "type III" corruptions. We need not assess the impact of leaking the secret s, since this leakage would allow the adversary to recover the entire chain of hash values. However, s is *only* used in the computation of the hash values. Therefore, the hash function's one-wayness ensures the secrecy of s.

Primitives. Symmetric key encryption is modeled by two symbols senc and sdec, whose meaning is captured by the following equation

$$\forall x, \forall k\ \mathsf{sdec}(k, \mathsf{senc}(k, x)) = x$$

The hash function H and the message authentication code function MAC are simply function symbols for which the absence of equation that decomposes $H(x)$ or $MAC(k, x)$ ensure the function's onewayness. That is, the only way for an adversary to build the term $MAC(k, x)$ $(H(x))$ is by knowing the symbols k and x (resp. x).

Other symbols. As usual in ProVerif, we use the symbol c for "broadcast channel" to model an insecure network. We also use symbols of constants to represent the cycle counter (CC), the session number (i) and the identities of the participants IDA, IDB.

Processes. We now describe the processes corresponding to the roles of Alice and Bob (i.e. initiator and responder respectively). The complete input file given to ProVerif can be found in appendix B.

– Process for participant A :

1. let $\mathsf{h1} = \mathsf{h}(s)$ in let $\mathsf{h2} = \mathsf{h}(\mathsf{h1})$ in let $\mathsf{h3} = \mathsf{h}(\mathsf{h2})$ in
2. new r1
3. let $m = \langle 1, \mathsf{CC}, \mathsf{i}, \mathsf{r1}, \mathsf{A}\rangle$
4. out(c, $\langle \mathsf{A}, \mathsf{i}, \mathsf{senc}(\mathsf{h3}, \langle m, \mathsf{h}(m)\rangle)\rangle$)
5. in(c, $\mathsf{msg}(\mathsf{B}, \mathsf{i}, x)$)
6. let $\langle m', y\rangle = \mathsf{sdec}(\mathsf{h3}, x)$ in
7. let $\langle 2, \mathsf{CC}, \mathsf{i}, \mathsf{r2}, \mathsf{r1}, \mathsf{B}\rangle = m'$ in let $\langle \mathbf{k_{se}}, \mathbf{k_{sa}}, \mathbf{k_{conf}}\rangle = \mathsf{mac}(\mathsf{h2}, \langle \mathsf{CC}, \mathsf{i}, \mathsf{r1}, \mathsf{r2}, \mathsf{h2}\rangle)$
8. if $y = \mathsf{mac}(\mathbf{k_{conf}}, m')$ then
9. out(c, $\langle \mathsf{A}, \mathsf{i}, \mathsf{mac}(\mathsf{h1}, \langle 3, \mathsf{A}, \mathsf{CC}, \mathsf{i}, \mathsf{r1}, \mathsf{r2}\rangle)\rangle$)
10. out(c, $\langle \mathsf{senc}(x, \mathbf{k_{se}}), \mathsf{senc}(s, \mathbf{k_{se}}), \mathsf{senc}(x, \mathbf{k_{sa}}), \mathsf{senc}(s, \mathbf{k_{sa}})\rangle$)

– Process for participant B :

1. let $\mathsf{h1} = \mathsf{h}(s)$ in let $\mathsf{h2} = \mathsf{h}(\mathsf{h1})$ in let $\mathsf{h3} = \mathsf{h}(\mathsf{h2})$ in
2. new r1
3. in(c, $\langle \mathsf{A}, \mathsf{i}, x\rangle$)
4. let $\langle m, y\rangle = \mathsf{sdec}(\mathsf{h3}, x)$
5. if $y = \mathsf{h}(m)$ then
6. let $\langle 1, \mathsf{CC}, \mathsf{i}, \mathsf{r1}, \mathsf{A}\rangle = m$ in let $\langle \mathbf{k_{se}}, \mathbf{k_{sa}}, \mathbf{k_{conf}}\rangle = \mathsf{mac}(\mathsf{h2}, \langle \mathsf{CC}, \mathsf{i}, \mathsf{r1}, \mathsf{r2}, \mathsf{h2}\rangle)$
7. let $m' = \langle 2, \mathsf{CC}, \mathsf{i}, \mathsf{r2}, \mathsf{r1}, \mathsf{B}\rangle$ in
8. out(c, $\langle \mathsf{B}, \mathsf{i}, \mathsf{senc}(\mathsf{h3}, \langle m', \mathsf{mac}(\mathbf{k_{conf}}, m')\rangle)\rangle$)
9. in(c,$\langle \mathsf{A}, \mathsf{i}, \mathsf{mac}(\mathsf{h1}, \langle 3, \mathsf{A}, \mathsf{CC}, \mathsf{i}, \mathsf{r1}, \mathsf{r2}\rangle)\rangle$)
10. out(c, $\langle \mathsf{senc}(x, \mathbf{k_{se}}), \mathsf{senc}(s, \mathbf{k_{se}}), \mathsf{senc}(x, \mathbf{k_{sa}}), \mathsf{senc}(s, \mathbf{k_{sa}})\rangle$)

Results. Type I corruptions do not allow for breaking the corresponding session. In the case of type II corruptions, as long as one of the three hash values involved in one session remains secret, the adversary cannot deduce *any* of the session keys. That is, the keys exchanged in the current, next, and previous sessions remain secret. In the case that all three hash values are leaked, then the current session is broken (i.e. the adversary learned secret2) while the other sessions remain safe. This result holds independently of whether the nonces are leaked or not. Thus, the only way to compromise a session would be to reveal all three hash values. However, this would not affect other sessions since future (resp. previous) values of the hash chain are made inaccessible by the secrecy of s (resp. the one-wayness of H).

4.3 Discussion

We will present the analysis of the protocol in its entirety. The main purpose is to demonstrate that the security of our protocol, which is more realistic and complete, can be formally analyzed. Our protocol can be represented like in the Figure 2. The arrows with 1 stands for the initialization step. The arrows marked with 2 represents the complete set of sessions during a cycle. The arrows marked with 3 illustrate the invocation of the renewing subprotocol. Finally, the arrow marked with 4 symbolizes the break point between two lifecycle and can stand for the renewing during the key derivation and the one-way property of the derivation.

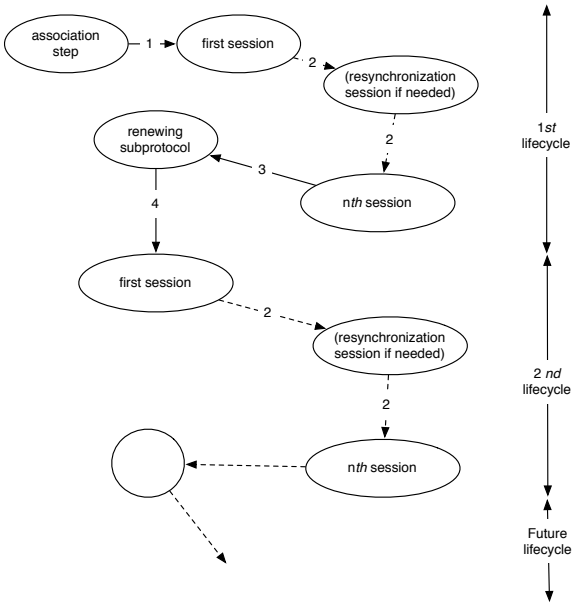

Fig. 2. The graphical representation of the protocol

We declare now that two lifecycle are independent thanks to its renewing subprotocol, because the one-way property and the use of random nonces produce a new secret that is strongly independent from the previous. This independence implies that our analysis can be restricted to one lifecycle.

If we examine a lifecycle, Alice and Bob will process $\lfloor \frac{n}{3} - 1 \rfloor$ sessions. The rest of the analyses are conducted with the use of formal methods, because they can scrutinize the security of single or multiple concurrent sessions of the main subprotocol.

Furthermore if the sessions are indeed independent, the security analysis can be reduced to single session execution. A session is independent from the others if it does not deliver information about past and future sessions. Proving the independence is done with the use of automated tools.

During a lifecycle, a session i has the most informations about the $(i-1)th$ and $(i+1)^{th}$ sessions. Proving this independence consists in proving that both consecutive sessions are independent. By induction, if the session $(i-1)th$ and i are independent and the session i and $(i+1)^{th}$ are independent, then by transitivity the sessions $(i+1)^{th}$ and $(i-1)^{th}$ are independent.

We have devised and made the analysis of parallel executions of three consecutive sessions with the tools AVISPA and ProVerif in sections 4.1 and 4.2 respectively. The analysis has not found any attack such as impersonation, replay, or man-in-the-middle attacks. We can consider then that the security of each session is equivalent to the security of one session. Furthermore, this independence proves that the protocol is backward and forward secure.

The construction of the chain (of hashed values) is an important aspect in the security of our subprotocols. In [8], the authors describe efficient and secure methods to compute hash chains for authentication. These methods are stronger than ours, but they make the assumption that a hashed value of the chain may be revealed after its use. In that case, the collection of the revealed values provide the ability to forge false chains of hashed values. We do not make such an assumption, because the elements of our hash chain are thrown after their use and therefore never revealed. Nonetheless, it remains possible to use the construction developed in [8] for our protocol.

5 Other security finding

If the underlying technology of implementation is based on very short range communication technologies (such as RFID or NFC technologies), it would be possible to detect cloned device thanks to the value of *CC*.

The cloned device will have access during a maximum of n sessions. After those n sessions, the key renewing will be made with one of the two devices (the legit or the cloned device). After the renewing, we have two possibilities. First, the legit user has made the renewing and then the cloned has not the new secret s and cannot process any session (the device should be cloned again). In the second case, the cloned device has made the renewing, but the user cannot process a session and then he detects the cloning.

This cloning detection is efficient only if eavesdropping is made really hard, because the cloned device could make the renewing by listening the renewing processed by the legit device. This security finding is limited to very short range communication technologies for that reason.

6 Future Works and Conclusion

We have developed a generic and efficient authenticated key agreement protocol for lightweight devices. This protocol provides automated key renewing, contributivity and security against nonce corruption. This protocol has been verified to be secure with AVISPA and ProVerif. Future studies will bring new

protocols with concrete cryptographic primitives for some specific applications. For instance, the usage of our authenticated key agreement protocol can be applied to create more robust authenticated distance bounding protocols for RFID technologies.

References

1. Aboba, B., Blunk, L., Vollbrecht, J., and Carlson, J. Extensible authentication protocol *(EAP)*. RFC 3748, June 2004.
2. Blanchet, B. Automatic verification of correspondences for security protocols. *Journal of Computer Security 17*, 4 (July 2009), 363–434.
3. Challal, Y., Bouabdallah, A., and Hinard, Y. Efficient multicast source authentication using layered hash-chaining scheme. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks* (Washington, DC, USA, 2004), LCN '04, IEEE Computer Society, pp. 411–412.
4. Choi, S. Denial-of-service resistant multicast authentication protocol with prediction hashing and one-way key chain. In *Proceedings of the Seventh IEEE International Symposium on Multimedia* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 701–706.
5. Diffie, W., and Hellman, M. E. New directions in cryptography. In *IEEE Transactions on Information Theory* (1976), vol. 22, pp. 644–654.
6. Dolev, D., and Yao, A. On the security of public key protocols. *Information Theory, IEEE Transactions on 29*, 2 (1983), 198–208.
7. Fluhrer, S., Mantin, I., and Shamir, A. Weaknesses in the key scheduling algorithm of rc4. In *Proceedings of the 4th Annual Workshop on Selected Areas of Cryptography* (2001), S. B. . Heidelberg, Ed., pp. 1–24.
8. Hu, Y.-C., Perrig, A., and Jakobsson, M. Efficient constructions for one-way hash chains. In *Applied Cryptography and Network Security* (New York, NY, June 2005).
9. Jeong, I., Katz, J., and Lee, D. One-round protocols for two-party authenticated key exchange. In *Applied Cryptography and Network Security*, M. Jakobsson, M. Yung, and J. Zhou, Eds., vol. 3089 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004, pp. 220–232.
10. Klein, A. Attacks on the rc4 stream cipher. *Des. Codes Cryptography 48* (September 2008), 269–286.
11. Krawczyk, H. Cryptographic extraction and key derivation: The hkdf scheme. In *Advances in Cryptology, CRYPTO*, T. Rabin, Ed., vol. 6223 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, pp. 631–648.
12. LaMacchia, B., Lauter, K., and Mityagin, A. Stronger security of authenticated key exchange. In *Proceedings of the 1st international conference on Provable security* (Berlin, Heidelberg, 2007), ProvSec'07, Springer-Verlag, pp. 1–16.
13. Lamport, L. Password authentication with insecure communication. *Communications of the ACM 24*, 11 (November 1981), 770–772.
14. Needham, R., and Schroeder, M. Using encryption for authentication in large networks of computers. *Communications of the ACM 21*, 12 (December 1978), 993–999.
15. Neuman, B. C., and Ts'o, T. Kerberos: An authentication service for computer networks. *IEEE Communications 32*, 9 (September 1994), 33–38.

16. Qachri, N., and Roggeman, Y. The flaws and critics about the security layer for the wimedia mac standard. In *30-th symposium on Information Theory in the Benelux* (may 2009), pp. 89–96.
17. Stubblefield, A., Ioannidis, J., and Rubin, A. D. Using the fluhrer, mantin, and shamir attack to break wep. Tech. Rep. TD-4ZCPZZ, AT&T Labs, 2001.
18. team, T. A. Avispa v1.1 user manual, June 2006.
19. team, T. A. Hlpsl tutorial: A beginner's guide to modelling and analysis internet security protocols, June 2006.
20. Wang, F., and Zhang, Y. A new provably secure authentication and key agreement mechanism for sip using certificateless public-key cryptography. In *2007 International Conference on Computational Intelligence and Security* (December 2007), IEEE, Ed., pp. 809–814.
21. Zou, X., Thukral, A., and Ramamurthy, B. An authenticated key agreement protocol for mobile ad hoc networks. In *Mobile Ad-hoc and Sensor Networks. Second International Conference, MSN 2006. Proceedings (Lecture Notes in Computer Science Vol. 4325)*. Springer-Verlag, January 2006.

Appendix A. HLPSL specification of the main subprotocol

```
%%% This is the Alice Role of the authenticated key agreement protocol
%%% where the input parameters are the main knowledge of Alice and Bob

role alice (A,B: agent,
   S: symmetric_key,
   Hash: hash_func,
   CC: nat,
   Sess: nat,
   SND,RCV : channel (dy))

played_by A def=
 local
  State: nat,
  Kconf: message,
  R1,R2: text,
  Authb: message,
  K1: symmetric_key,
  K2: symmetric_key,
  K3: symmetric_key,
  X : hash(message.hash(message.nat.nat.nat.text.text.agent)) %%% expression of the
                                                              %%% knowledge on the
                                                              %%% format  used to
                                                              %%% compute  the message
                                                              %%% authentication code


 %%% creation of the 3 keys from the shared secret
 init
  State :=1 /\
  K1:=Hash(Hash(Hash(Hash(S).S).S).S) /\
  K2:=Hash(Hash(Hash(S).S).S) /\
  K3:=Hash(Hash(S).S)

 transition

 %%% First message that launchs the key agreement with a fresh Nonce R1

 1. State=0 /\ RCV(start) =|>
     State':= 2 /\ R1':=new() /\ SND(A.Sess.{1.CC.Sess.R1'.A.Hash(1.CC.Sess.R1'.A)}_K1)
```

```
 %%% Answer of Bob with R2 and the MAC that confirms that the key derived by Alice is
 %%% the same than Bob

 2. State=2 /\ RCV(B.Sess.{2.CC.Sess.R2'.R1.B.X'}_K1)
       /\ X' = Hash(xor(Hash(K2.Hash(K2.CC.Sess.R1.R2'.K3)) ,23644).Hash(xor(Hash(K2.
          Hash(K2.CC.Sess.R1.R2'.K3)),13878).2.CC.Sess.R2'.R1.B))
          =|> State':=4 /\ Kconf':=Hash(K2.Hash(K2.CC.Sess.R1.R2'.K3))
            /\ SND(Hash(xor(Kconf',23644).Hash(xor(Kconf',13878).3.A.CC.Sess.R1.R2)) )
            /\ witness(A,B,alice_bob_R2,R2')
            /\ witness(A,B,derive_ab_ks,Kconf')
            /\ request(A,B,bob_alice_R1,R1)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% This is the role played by Bob in the system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role bob (
 A,B: agent,
 S: symmetric_key,
 Hash: hash_func,
 CC: nat,
 Sess: nat,
 SND,RCV : channel (dy))

played_by B def=
 local
  State: nat,
  Kconf: message,
  R1,R2: text,
  Autha: message,
  K1: symmetric_key,
  K2: symmetric_key,
  K3: symmetric_key,
  X3: hash(nat.nat.nat.text.agent),
  X2: hash(message.hash(message.nat.agent.nat.nat.text.text))


 init
  State :=1 /\
  K1:=Hash(Hash(Hash(Hash(S).S).S).S) /\
  K2:=Hash(Hash(Hash(S).S).S) /\
  K3:=Hash(Hash(S).S)

 transition

 %%% Reception of the message of Alice that wants make a key agreement
 %%% based on the secret shared keys, on the fresh nonce of Alice and on
 %%% the nonce chosen by Bob

 1. State=1 /\ RCV(A.Sess.{1.CC.Sess.R1'.A.X3'}_K1) /\ Hash(1.CC.Sess.R1'.A) = X3'
     =|>  State':= 3 /\ R2':=new()  /\ Kconf':= Hash(xor(K2,23644).Hash(xor(K2,13878).
          CC.Sess.R1.R2'.K3)) /\ SND(B.Sess.{2.CC.Sess.R2'.R1'.B.Hash(xor(Kconf',
          23644).Hash(xor(Kconf',13878).2.CC.Sess.R2'.R1.B))}_K1) /\ secret(Kconf',
          kconf,{A,B}) /\ witness(A,B,bob_alice_R1,R1')

 %%% Last received message where Bob checks the good ending of the protocol and that
 %%% the authentication is well achieved

 2. State = 3 /\ RCV(X2') /\ Hash(xor(Kconf,23644).Hash(xor(Kconf,13878).3.A.CC.Sess.
      R1.R2)) = X2'
      =|>  State':=5  /\ request(A,B,alice_bob_R2,R2) /\ request(A,B,derive_ab_ks,
      Kconf)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% This role makes the composition of the roles of Alice and Bob and describes a
%%% session of the  protocol
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session(A,B : agent,
    S: symmetric_key,
    Hash : hash_func,
    CC: nat,
    Sess: nat)

def=
 %%% Define of the channel of communications
 local SA, SB, RA, RB: channel (dy)

composition
 alice (A, B, S, Hash, CC, Sess, SA, RA)
 /\ bob (A, B, S, Hash, CC, Sess, SB, RB)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment()
def=

const
  kconf, bob_alice_R1, alice_bob_R2, derive_ab_ks : protocol_id,
  kab, kai, kib     : symmetric_key,
  a, b : agent,
  h  : hash_func


intruder_knowledge = {a, b, h, kai, kib}  %%% it defines what knows an intruder i

%% The test involves 4 parallel sessions

composition

 %% Two legitimates sessions between Alice and Bob are executed with consecutive number
 %% and keys of session

 session(a, b, kab, h, 1, 1)  /\ session (a,b, h(h(h(h(kab).kab).kab.).kab), h, 1,2)

 %% Two sessions where the intruder plays one of the two role in a protocol session

 /\ session(a, i, kai, h, 1, 2)  /\ session(i, b, kib, h, 1, 3)

end role

 %%% The goal section describes the security goals that must achieve the protocol to
 %%% confirm the mutual authentication and to ensure the secrecy of the derived keys

goal

 secrecy_of kconf
 authentication_on bob_alice_R1
 authentication_on alice_bob_R2
 authentication_on derive_ab_ks

end goal

environment()
```

Appendix B. ProVerif specification of the main subprotocol

```
(* T Y P E S *)
type N. (*nonce*)
type M. (*message*)
type I. (*identity*)
type R. (*round*)
type O. (*other*)

(* S Y M B O L S *)
const RND1, RND2, RND3:R.
const IDA, IDB: I.
const CC, i1, i2, i3: O.


(* P R I M I T I V E S *)
fun h(bitstring): bitstring.
fun mac(bitstring, bitstring): bitstring.
fun senc(bitstring, bitstring): bitstring. (*/!\ this is authenticated encryption*)
reduc forall k: bitstring, x: bitstring; sdec(k, senc(k, x)) = x.


(* A D V E R S A R Y *)
free c: channel.
set attacker = active.
free s, secret1, secret2, secret3: bitstring [private].
query attacker (secret1).
query attacker (secret2).
query attacker (secret3).


(* P R O C E S S *)

let processA1 = new rA:N;
        let k2 = h(s) in
    let k1 = h((k2, s)) in
    let k0 = h((k1, s)) in
    let m = (RND1, CC, i1, rA, IDA) in
    let x = ( m, h(m) ) in
    out(c, (IDA, i1, senc(k0 , x)));
    in( c, (= IDB, = i1, y:bitstring) );
    let (mp:bitstring, macmp:bitstring) = sdec(k0 , y) in
        let (= RND2, = CC, = i1, rB :N, = rA, = IDB) = mp in
        let KEYS = mac(k2 , (CC, i1, rA, rB , k1 )) in
        if macmp = mac(KEYS , mp) then
          out(c, (IDA, i1, mac(KEYS , (RND3, IDA, CC, i1, rA, rB ))));
        out(c, senc(KEYS, secret1)).

let processB1 = new rB:N;
        let k2 = h(s) in
        let k1 = h((k2, s)) in
        let k0 = h((k1, s)) in
        in( c, (= IDA, = i1, y:bitstring));
        let (m:bitstring, = h(m)) = sdec(k0 , y) in
        let (= RND1, = CC, = i1, rA:N, = IDA) = m in
        let KEYS = mac(k2 , (CC, i1, rA, rB , k1 )) in
        let mp = (RND2, CC, i1, rB , rA, IDB) in
        let xp = (mp, mac(KEYS , mp)) in
        out(c, (IDB, i1, senc(k0 , xp)));
        in( c, (= IDA, = i1, (= RND3, = IDA, = CC, = i1, = rA, = rB )) );
        out(c, senc(KEYS, secret1)).

let processA2 = new rA:N;
        let tmp1 = h((h(s), s)) in
        let tmp2 = h((tmp1, s)) in
        let k2 = h((tmp2, s)) in
    let k1 = h((k2, s)) in
    let k0 = h((k1, s)) in
        let m = (RND1, CC, i2, rA, IDA) in
        let x = ( m, h(m) ) in
```

```
        out(c, (IDA, i2, senc(k0 , x )));
        in( c, (= IDB, = i2, y:bitstring) );
        let (mp:bitstring, macmp:bitstring) = sdec(k0 , y) in
        let (= RND2, = CC, = i2, rB :N, = rA, = IDB) = mp in
        let KEYS = mac(k2 , (CC, i2, rA, rB , k1 )) in
        if macmp = mac(KEYS , mp) then
        out(c, (IDA, i2, mac(KEYS , (RND3, IDA, CC, i2, rA, rB ))));
        out(c, senc(KEYS , secret2)).

let processB2 = new rB:N;
        let tmp1 = h((h(s), s)) in
        let tmp2 = h((tmp1, s)) in
        let k2 = h((tmp2, s)) in
   let k1 = h((k2, s)) in
   let k0 = h((k1, s)) in
        out(c, s);
        in( c, (= IDA, = i2, y:bitstring));
        let (m:bitstring, = h(m)) = sdec(k0 , y) in
        let (= RND1, = CC, = i2, rA:N, = IDA) = m in
        let KEYS = mac(k2 , (CC, i2, rA, rB , k1 )) in
        let mp = (RND2, CC, i2, rB , rA, IDB) in
        let xp = (mp, mac(KEYS , mp)) in
        out(c, (IDB, i2, senc(k0 , xp)));
        in( c, (= IDA, = i2, (= RND3, = IDA, = CC, = i2, = rA, = rB )) );
        out(c, senc(KEYS , secret2)).

let processA3 = new rA:N;
        let tmp1 = h((h(s), s)) in
        let tmp2 = h((tmp1, s)) in
        let tmp3 = h((tmp2, s)) in
        let tmp4 = h((tmp3, s)) in
        let tmp5 = h((tmp4, s)) in
        let k2 = h((tmp5, s)) in
   let k1 = h((k2, s)) in
   let k0 = h((k1, s)) in
        let m = (RND1, CC, i3, rA, IDA) in
        let x = ( m, h(m) ) in
        out(c, (IDA, i3, senc(k0 , x )));
        in( c, (= IDB, = i3, y:bitstring) );
        let (mp:bitstring, macmp:bitstring) = sdec(k0 , y) in
        let (= RND2, = CC, = i3, rB :N, = rA, = IDB) = mp in
        let KEYS = mac(k2 , (CC, i3, rA, rB , k1 )) in
        if macmp = mac(KEYS , mp) then
        out(c, (IDA, i3, mac(KEYS , (RND3, IDA, CC, i3, rA, rB ))));
        out(c, senc(KEYS , secret3)).

let processB3 = new rB:N;
        let tmp1 = h((h(s), s)) in
        let tmp2 = h((tmp1, s)) in
        let tmp3 = h((tmp2, s)) in
        let tmp4 = h((tmp3, s)) in
        let tmp5 = h((tmp4, s)) in
        let k2 = h((tmp5, s)) in
   let k1 = h((k2, s)) in
   let k0 = h((k1, s)) in
        in( c, (= IDA, = i3, y:bitstring));
        let (m:bitstring, = h(m)) = sdec(k0 , y) in
        let (= RND1, = CC, = i3, rA:N, = IDA) = m in
        let KEYS = mac(k2 , (CC, i3, rA, rB , k1 )) in
        let mp = (RND2, CC, i3, rB , rA, IDB) in
        let xp = (mp, mac(KEYS , mp)) in
        out(c, (IDB, i3, senc(k0 , xp)));
        in( c, (= IDA, = i3, (= RND3, = IDA, = CC, = i3, = rA, = rB )) );
        out(c, senc(KEYS , secret3)).


process (!processA1 |!processB1 )
        | (!processA2 |!processB2 )
```

| (!$processA3$ |!$processB3$)