

# Secure device pairing under realistic conditions

Olivier Courtay, Olivier Heen, Mohamed Karroumi, and Alain Durand

Thomson R&D France, Security Lab  
olivier.courtay@thomson.net  
olivier.heen@thomson.net  
mohamed.karroumi@thomson.net  
alain.durand@thomson.net

**Abstract.** Creating a trust relationship between two devices is commonly designated as pairing protocol. This practice is essential for the security of dynamic networks as it protects against rogue device insertion. If the pairing itself is not secure all afterward security will be compromised. All existing methods require either a pre-existing secure channel between devices or a common security context (e.g. Public Key Infrastructure) or the handling of cryptographic information by the user. All these requirements are rarely achieved in practice. We propose a solution where these requirements are relaxed. In particular, the user is not required to memorize type-in or even check any cryptographic information. Therefore, this solution is suitable for devices with only LEDs and a button such as WiFi access points, modems and small portable devices. Keywords: Dynamic Network Security, Device Pairing, Key Agreement.

## 1 Introduction

Pairing two devices consists in setting a mutual trust relationship. Typically the trust relationship is established when devices are able to share a common secret. This shared secret can be used as a basis for secure communications.

A remaining problem is how to achieve that where the devices can communicate via a given network, are on the local control of the user and have no common information?

A naive solution would be to use a Diffie-Hellman (DH) key agreement between pairs, via a broadcasted network communication. However, if used without a robust certification mechanism, DH protocol is vulnerable to man-in-the-middle attacks. Certification mechanisms are generally based on a public key infrastructure, but in that case a security context is shared by the devices.

In this article, we provide a secure pairing solution for simple devices, with no secure channel, no shared security context and user friendliness:

**Simple devices:** the solution should be adapted for a large choice of devices like personal computers, WiFi access points, printers, portable devices. . . . Devices may have limited interaction abilities: LEDs and one button instead of screen and keyboard.

**No secure channel:** the channel used to set-up the trust relationship is not dedicated . Message injection, interception or eavesdropping is possible at any time.

**No shared security context:** no specific security configuration or architecture are required before the beginning of the pairing operation. Devices and attacker have the same initial information. In particular, devices do not already share a secret or trust in any certification authority. . .

**User friendliness:** user follows straightforward instruction, and performs clicks according to LED behavior. It is not required from the user to read or type any password, cryptographic value nor to understand devices internal states.

No current solution fully meets these requirements. For instance, time-limited channel like resurrecting duckling [9] provides user friendly solution using neither screen nor keyboard. Nevertheless, but during the initialization step a secure channel is required (e.g. an USB wire between the two devices). Location-limited channel (e.g. IrDA link) provides limited secure solution that does not fits local area network. Other solutions exist for devices possessing screen and keyboard [1], those use for example *random-art* comparison or *small-words* comparison [7]. Also, these solutions need many iterations from user, such as reading, comparing or writing cryptographic information. This could be annoying for the user and very simple devices do not allow such advanced interactions. Recent solutions like [3] or [5] meets the requirements but are vulnerable to man-in-the-middle attacks, see [6] for a detailed analysis.

In section 2 give some notations, hypothesis and describe our solution. In section 3 we discuss correctness, security of the solution. Then, in section 4, we suggest implementation choices and address a special case for wireless networks.

## 2 Our proposal

In our solution, designed for Local Area Networks or LAN, exchanged identifiers will be devices public keys. The protocol is based on cryptographic network messages, user controls and actions that guarantee the key obtained by each device is the public key of the expected device. The protocol is a key agreement protocol that resist to man-in-the-middle attacks. This is achieved thanks to authoritative user controls and the counting of active devices that are present on the network during the pairing operation.

The protocol is designed for device with following characteristics:

- Allow one user action. This action can be typically to press a button (eventually emulated on a screen).
- Display a small number of different states, for instance using LEDs, LCD screens or computer screens. Only two LEDs are actually required.
- Attackers can not perform physical actions on the device, such as pressing a button, unplugging it, or physically extracting keys. . .

## 2.1 Notations

Each device has eight different states. States are denoted:  $I$  for Initial,  $L$  for Listen,  $F$  for Final,  $E$  for Error,  $C$  for Client,  $P$  for Pre-final,  $W$  for Waiting and  $S$  for Selected.

State transitions are triggered either by user actions or by messages. User actions can be either: pressing device  $x$  button, denoted  $action(x)$ , or waiting for device  $x$  to enter in state  $t$ , denoted  $wait(x, t)$ . Messages have one type and the public key of the sender and can contain payload with cryptographic value. Messages types are *Challenge*, *Challenge – Response*, *Response*, *Error*. Each device may store sent or received message. Each received or sent message are denoted with name that depends of the name of the source(not identified) and of the context. For example, a *Challenge* message sent by the device  $a$  is called  $Ch(a)$  for  $a$ . The same message received by device  $b$  is noted  $Ch(a_1)$ ; the generated response is noted  $ChRe(b, Ch(a_1))$ .  $Pub_a$  is device  $a$  public key.

## 2.2 Network assumptions

The protocol is designed for networks with the **no hidden selective message destruction** property: after a reasonable delay, a sender detects that messages are not correctly sent or a receiver detects messages are not received.

If no selective message destruction is possible on the network, the restriction holds.

Examples of network where such attacker restrictions reasonably hold are: IEEE 1394 (FireWire) networks, USB direct connection, flat 802.3 networks possibly with hubs but without router switches (or only with trusted router switches). The case WiFi will be to study in 4.2 and we will show that the WiFi network are adapted thanks to an additional mechanism.

## 2.3 Cryptographic assumptions

Each device owns its public/private key pair, and keeps the private part confidential.

Cryptographic protocols involving message exchanges require precise definition of both the messages to be exchanged and the actions to be taken by each device. We give here the characteristics needed by our solution:

**Message link:** the protocol must provide devices with means to verify that a message is linked to a former message.

**Message authentication:** the protocol must provide devices with means to verify that a *Response* message was created using a private key associated to a public key received in a former *Challenge* message. The same property should also apply to *Challenge – Response* messages.

**Replay protection:** the protocol should provide means to prevent message replay attacks.

**Private keys confidentiality:** the protocol should provide means to protect private keys confidentiality. [10],[11].

## 2.4 User instructions

Without loss of generality, we always choose  $a$  to be the device where the first  $action()$  is performed. During pairing of devices  $a$  and  $b$ , the user must follow instructions in this order:

- $action(a)$ , the user performs an action on device  $a$ .
- $wait(a, W)$ , the user waits device  $a$  to be in state  $W$ .
- $action(b)$ , the user performs an action on the second selected device ( $b$ ).
- $wait(b, P)$ , the user waits device  $b$  to be in state  $P$ .
- $action(b)$ , the user performs a second action on device  $b$ .
- $action(a)$ , the user performs a second action on device  $a$ .

If a device never enters in the expected state, then the user shall do nothing. Users must strictly follow these instructions to obtain a high level of security. However, this is one of the requirements on users. Users do not have to understand the internal protocol. We will see in section 4 that check state and perform action can be very simple for user.

## 2.5 State machine

In paragraph 2.5 we describe the protocol in the case of the pairing of two devices  $a$  and  $b$ , both starting in state  $I$ .

A simplified finite state machine diagram is represented in 1.

We describe device internal behavior for each state, according to the received message or the user action.

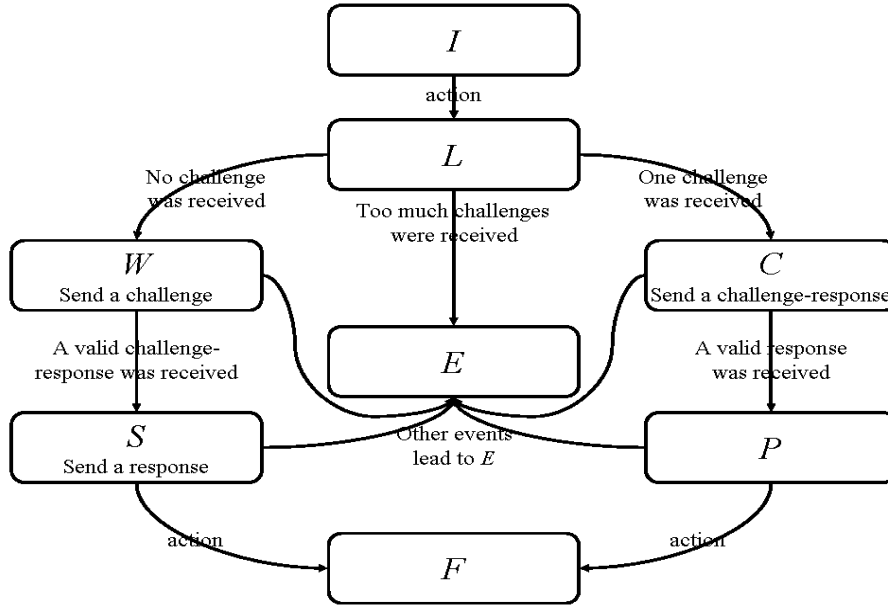
**State  $I$**  The device ignores all events, except  $action(local)$  that moves the device to state  $L$ .

**State  $L$**  The device is listening to signals from other devices during time  $T_1$ . Then, it behaves as follows:

- If exactly one *Challenge* message  $Ch(a_1)$  is received.  $Ch(a_1)$  contains a cryptographic challenge. The device memorizes this message then move to the state  $C$ .
- If no message is received: the device goes to the  $W$  state.
- If other event occurred (including user action, or reception of other messages): the device moves to  $E$  state.

**State  $C$**  The device builds a *Challenge–Response* message  $ChRe(local, Ch(a_1))$  containing a cryptographic response to the challenge included in message  $Ch(a_1)$  and a cryptographic challenge, and sends it on the network. Behavior in this state is:

- If it receives  $Ch(a_1)$  again,  $ChRe(local, Ch(a_1))$  is resent on the network.
- If it receives a *Response* message  $Re(a_2, ChRe(b_1, Ch(a_4)))$ .
  - If the response included in  $Re(a_2, ChRe(b_1, Ch(a_4)))$  is a valid cryptographic response of the challenge included in  $ChRe(local, chlg(a_1))$ , the device goes to state  $P$ .



**Fig. 1.** Simplified state machine. For readability, notations such as  $Ch(x)$  are simplified into  $Ch$ .

- If not, a *Error* message  $Err$  is prepared.  $Err$  contains a cryptographic response to the challenge included in message  $Ch(a_1)$ . Then the device moves to state  $E$ .
- If other event occurred (included a  $time(T_3)$ ): an authenticated *Error* message is prepared. It contains a cryptographic response to the challenge included in message  $Ch(a_1)$ . The authenticated *Error* message avoids unwanted final state (see end of subsection 3.1). Then the device moves to state  $E$ .

**State  $P$**  Behavior in this state is:

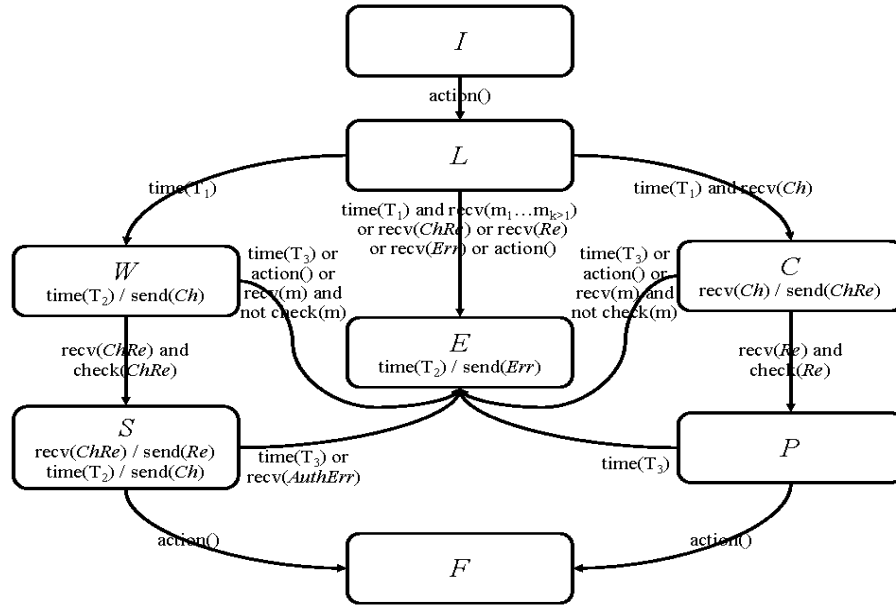
- If the user performs  $action(local)$ , the device goes to state  $F$ .
- If after a  $time(T_3)$ , the device moves to state  $E$ .
- Other events are ignored.

**State  $W$**  In this state, the device regularly (every  $T_2$  time, with an  $N$  let  $T_1 > N * T_2$ ) broadcasts a  $Ch(local)$  message. Behavior in this state is:

- It receives the *Challenge – Response* message  $ChRe(b_1, Ch(a_3))$ .
  - if  $ChRe(b_1, Ch(a_3))$  contains a valid cryptographic response, the device goes to state  $S$ .
  - if not, the device goes to state  $E$ .
- If other event occurred or after a  $time(T_3)$ : Then the device moves to state  $E$ .

- State  $S$**  When it enters in this state, the device sends on the network a *Response* message  $Re(local, ChRe(b_1, Ch(a_3)))$ . The device continues to regularly broadcast the  $Ch(local)$  message. Behavior in this state is:
- If it receives again  $ChRe(b_1, Ch(a_3))$ , the device resends  $Re(local, ChRe(b_1, Ch(a_3)))$ .
  - If it receives a *Error* message.
    - If the message contains a valid cryptographic response, the device goes to state  $E$ .
    - If not this message is ignored.
  - If the user performs  $action(local)$ , the device goes to state  $F$ .
  - After a  $time(T_3)$ , the device moves to state  $E$ .
- State  $E$**  If no *Err* message is already built, a *Err* message of type *Error* is prepared. In this state, the device regularly broadcasts *Err*.
- State  $F$**  The device does nothing

The figure 2 is the complete finite state machine diagram representation.



**Fig. 2.** State machine.

### 3 Protocol analysis

In this section, we discuss protocol properties. The protocol is correct (it works as specified) and secure (attacks cannot lead to unspecified behavior) if properties 1 and 2 are verified.

**Property 1** *if devices  $a$  and  $b$  execute the protocol until they reach the final state  $F$ , then device  $a$  has accepted a unique public key  $Pub_b$  and device  $b$  has accepted a unique public key  $Pub_a$ .*

**Property 2** *if the user respects instruction of the protocol with two devices  $a$  and  $b$  then both devices go either from initial state  $I$  to final state  $F$ , either from initial state  $I$  to error state  $E$ .*

More precisely, operational conditions can be described according to the following hypotheses:

**UI(a,b):** the user obeys user instructions and activate devices  $a$  and  $b$ .

**perfect connectivity:** there is no message loss.

**n connectivity:** If a message is sent  $n$  times at most, it finally arrives.

**no connectivity:** Either each message is correctly delivered, or no message is received or emitted (the canonical example is a wire: plug or unplug).

**no bad message:** All messages are compliant to the protocol, there are no replayed messages or forged messages.

The analysis of the protocol follows this strategy: first, we prove that the protocol meets the requested properties for a secure device pairing under ideal conditions. Second, we gradually relax some of the hypotheses and prove that the security and the correctness of the protocol still hold.

### 3.1 Proof under ideal conditions

In this section, we prove that our protocol is correct and secure under ideal conditions. Ideal conditions hold when **UI(a,b)**, **perfect connectivity** and **no bad message**.

A successful scenario including message exchanges and user actions is shown on 3.

**Lemma 1.** *An activated device eventually finishes in a stable state  $F$  or  $E$ .*

A simple analysis of the transition diagram shows that the protocol does not fall into continuous looping states, and that all states (except state  $E$  and  $F$ ) have timeout transitions. Thus, all activated devices are in stable states that are  $F$  or  $E$  after some time (after time  $T_1 + 2 * T_3$ ).

When both devices arrive in a stable state, only three cases are possible:

- $a$  and  $b$  are both in state  $F$
- $a$  and  $b$  are both in state  $E$
- One of them is in state  $F$  and the other is in state  $E$

**Lemma 2.** *If the first activated device finishes in state  $F$ , it necessarily goes through state  $W$ .*

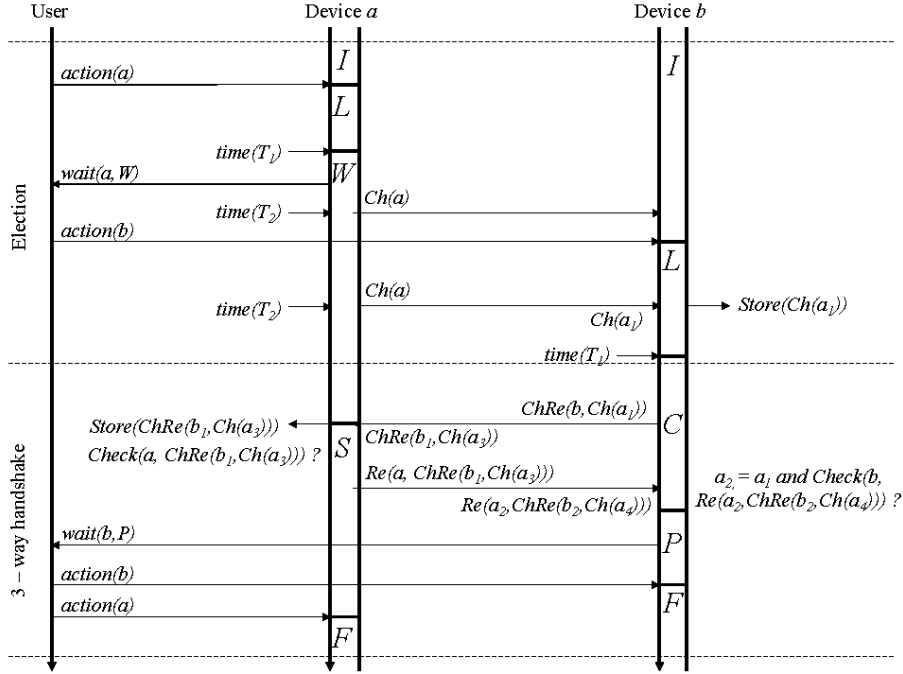


Fig. 3. Sequence diagram.

As the device finishes in state  $F$ , it never goes to state  $E$  (there is no path from  $E$  to  $F$ ). When the user activates the first device, the device goes to state  $L$ . Then, it goes to state  $W$  or  $C$ . If we suppose that it goes to state  $C$ , it necessarily goes then to state  $P$ . From state  $P$ , there is no path to go in state  $W$ . As the user obeys instructions, he is waiting for the device to be in state  $W$  before performing any action. Time  $T_3$  will thus elapse without any action and the device will go to state  $E$  what contradicts the hypothesis.

**Lemma 3.** *If the second activated device finishes in state  $F$ , it necessarily goes through state  $C$ .*

The second activated device goes in the state  $L$  and listens to packets from network during  $T_1$ . When the second device is activated, the first device is in state  $W$ ,  $S$  or  $E$  (it can not be in the state  $F$  because the user does not perform the action until the second device is in state  $P$ ). In each of these states, the first device broadcasts regularly a message (a *Challenge* or *Error* message) thus, the second device in the  $L$  state will receive one of these messages. The device can not go to state  $W$ , thus if the device finishes in state  $F$ , it necessarily went through state  $C$ . We have proved lemma 3.

By conjunction of lemma 2 and lemma 3 we have that if two devices go from state  $I$  to state  $F$ , the first activated goes through state  $W$  and the other goes through state  $C$ .

*Proof (Property 1).*

We suppose both devices reach state  $F$ .

Device public keys are embedded in only two messages: *Challenge* and *Challenge – Response*. The second activated device necessarily received at least one message *Challenge* (from lemma 3). It can not have received several messages as it would have gone from state  $L$  to state  $E$ . Furthermore, the received message necessarily comes from the first activated device (message is emitted in state  $W$  that can be reached only by first activated device, see lemma 2). Thus, the second activated device knows the first activated device public key.

As regards the first activated device, it necessarily received a message *Challenge – Response* (from lemma 2, the device went through state  $S$ ). If this message was not received from the second device, the *Response* message emitted in state  $S$  can never be correctly verified by the second device in state  $C$ . This means that the second device finishes in state  $E$ , what contradicts the hypothesis. Thus, the first activated device necessarily knows second device public key.

So we have proved the property 1: If two devices go from state  $I$  to  $F$  state, they have mutually got their public key.

*Proof (Property 2).*

We have proved that there are only two final stable state  $E$  and  $F$ . We will prove that  $(a_E, b_F)$  or  $(a_F, b_E)$  possibility never happened.

As the user obeys the instructions, we know that the first activated device goes to state  $F$  after second activated device. Thus, if, at the end of the protocol, one device is in state  $F$  while the other is in state  $E$ , the one in state  $E$  is necessarily the first activated one. If the second device is in state  $F$  this means that the user necessarily performed an action on second device when it was in state  $P$ . The state  $P$  could only be reached if the first device emitted message *Response* when it was in state  $S$ . As the user obeys instructions, he will perform an action on first device within time  $T_3$ . At that time, if the first device is state  $E$ , it is because it received an authenticated *Error* message sent by the second device. This cannot happen as second device is in state  $P$ . Thus, the user action will cause the first device to go to state  $F$ .

**Remark concerning authenticated *Error*:** if devices are in state  $(a_S, b_P)$  an attacker may send a fake *Error* message. If  $a$  performs no verification, then the whole system will eventually reach the state  $(a_E, b_F)$ . The authenticated *Error* prevents this attack.

So we have proved property 2: If two devices  $a$  and  $b$  are activated by a user, two final states only are possible:  $(a_F, b_F)$  or  $(a_E, b_E)$ . Our protocol verifies properties 1 and 2.

### 3.2 Network security

We claim that the protocol is always secure and correct even if hypothesis **no bad message** is relaxed. In other words, the protocol is valid even if the attacker has the possibility to build and send messages over the network.

For each state, we now show that a forged or replayed message never leads to a situation where a device eventually enters state  $F$ .

**State  $L$ :** if an attacker sends a message, there are two cases.

**It is not a  $Challenge$ :** if the message is not a  $Challenge$  message, the device goes to state  $E$ , so no attack is possible.

**It is a  $Challenge$ :** if the message is a  $Challenge$  message thus the device goes in state  $C$  and two cases are possible:

**First activated device:** if this device was the first activated device by the user. The user waits for the device to in state  $W$ . This case never happens since, in state  $C$  or  $W$ , the device goes in state  $E$  after some time (time  $2 * T_3$ ).

**Second activated device:** if the device was the second activated device by the user. The device listens to all message during  $T_1$ . Simultaneously, the first activated device sends  $Challenge$  message. Thus, there are two cases:

**Same  $Challenge$ :** if the  $Challenge$  message sent by the attacker is the same as the  $Challenge$  message sent by first activated device, the second device ignores the replayed message.

**Different  $Challenge$ :** if the  $Challenge$  message is different, the device goes to state  $E$ .

**State  $C$ :** only a cryptographically valid  $Response$  message does not result in the device to transit to state  $E$ . in 2.3, we assumed that the attacker was unable to forge such message.

**State  $S$ :** only authenticated  $Error$  message changes the state of the device. As in state  $C$  case, the attacker can not inject a cryptographically valid message.

**State  $W$ :** the device is the first activated device by the user. It receives a  $Challenge-Response$  message and goes to state  $S$ . But the user will activate a second device. There are two cases.

**One  $Challenge$ :** when the second device is activated, it goes in state  $L$ . If it receives only one  $Challenge$  message during the duration  $T_1$  of this state, the message has been sent by the first device (or it was a replayed message by the attacker originally sent by the first device). Thus, the second device goes to state  $C$  and sends a  $Challenge-Response$  message. But, the first device is in state  $S$ , thus the  $Challenge-Response$  message are ignored. Thus, the second device never receives a cryptographically valid  $Response$  message and never goes to state  $P$ . After the timeout, the second device goes to state  $E$  and the user never see state  $P$  on the second device and thus never do  $action()$  on the first device.

**More than one  $Challenge$ :** if the second device receives several distinct  $Challenge$  messages, it goes in state  $E$ .

**State  $I$ ,  $P$ ,  $E$  and  $F$ :** messages are ignored.

We have proved that the protocol is correct and secure under the **UI(a,b)** and **perfect connectivity** hypotheses.

### 3.3 Relaxing connectivity hypotheses

In this section, we choose to progressively relax hypothesis **perfect connectivity**, until arriving to the assumption of a realistic network, as described in 2.2.

**Relaxing perfect connectivity** Let us examine what happens when the network does not offer property **perfect connectivity**, but offers property **n connectivity** instead. Property **n connectivity** states that messages may be retransmit many times before they arrive.

The diagram state has been built in such a way that message repetition does not modify devices behavior. Retransmission frequency should be tuned according to timeout values  $T_1, T_2, T_3$  used in the protocol.

Three types of messages loss may happen:

*Challenge*: Since this message is regularly broadcasts, it will eventually arrive (this is the **n connectivity** hypotheses).

*Challenge – Response* or *Response*: this message is sent by a device in state  $C$  and it waits a cryptographically valid *Response* message. The peer device is in state  $W$  or  $S$  and in these states, a *Challenge* message is regularly broadcasted. Thus, the lost *Challenge – Response* message is also resent. With the **n connectivity** hypotheses, the *Challenge – Response* message is received.

*Error*: these messages are regularly sent and always trigger device transition to state  $E$ .

**Relaxing n connectivity** Under realistic conditions, it may happen that some messages never reach their destination. This corresponds to hypothesis **no connectivity**: total loss of connectivity at some time during the protocol. A careful examination shows that situations like  $(a_E, b_F)$  or  $(a_F, b_E)$  can not happen. We only give justification instead of per case verification. If the first device is in state  $L$ ,  $W$  or  $C$  and does not receive any message, timeouts eventually lead the device to state  $E$ , and thus the device never emits any response message. Thus, timeout leads the other device to state  $E$  as well. In all the other cases, including  $(a_S, b_P)$ , user action is required to trigger state transition. Under hypothesis **UI(a,b)**, these transitions eventually lead to  $(a_F, b_F)$  (cf. 3.1).

**Allowing selective message destruction** We have shown that our protocol is correct and secure if hypothesis **no connectivity** is valid. This hypothesis is not valid in real network LAN where an attacker can intercept messages without any device to detect this dropping. This is the only difference between reality and a network that would respect hypothesis **no connectivity**. Thus, if our protocol

resists even when selective message destruction is possible, we can assume that our protocol can be used in networks that respect properties described in 2.2.

A dropping detection mechanism can be added to our protocol. The behavior of a device when it detects a dropping during the protocol is to go in state  $E$ . This dropping can be a normal collision or an attacker that tries to do a selective message destruction attack. An example of mechanism of detection of this kind of attack is described in 4.2.

Consequently, our protocol is secure and correct if the user obeys instructions and if selective message destruction can be detected on the underlying network.

## 4 Implementation and specific network issues

### 4.1 Implementation

**LED implementation:** assumptions on device characteristics are described in 2. One assumption is that device may only have LEDs. Devices states can be easily displayed using LEDs, so that the user easily identifies each situation before performing an action or waiting. For example, on the second device, the user shall check that the device is in  $C$ ,  $P$  branch and not in the  $W$ ,  $S$  branch. Thus, state  $P$  should be displayed differently from other states.

The following is a possible implementation of the protocol on a simple device with one button, one green LED and one red LED:

**Device is in state  $I$ ,  $L$ ,  $C$ :** no active LED.

**Device is in state  $W$ ,  $S$ :** the green LED blinks slowly.

**Device is in state  $P$ :** the green LED blinks quickly.

**Device is in state  $F$ :** the green LED is on.

**Device is in state  $E$ :** the red LED is on.

After having reached state  $F$  or  $E$ , the device returns after some timeout in state  $I$ .

**Cryptographic implementation:** The cryptographic part of the pairing is implemented with a straightforward variant of the Station-to-Station key agreement protocol described in [12]. A correct application of the protocol requires that:

- Each device owns an RSA key pair or is able to generate them. This is the case in the context of secure pairing.
- Each device has access to authentic copies of the other device public key. This is the case in the context of secure pairing thanks to the counting of active devices, as described in section 2.

Let  $g$  denote a Diffie-Hellman generator of a cyclic group of order  $p$  (where  $p$  is large prime),  $x$  and  $y$  denote nonces and  $||$  represents the concatenation. Let  $RSA(K, M)$  denote encryption of the message  $M$  using  $RSA$  algorithm with

asymmetric key  $K$ . Let  $AES(K, M)$  denote encryption of the message  $M$  using  $AES$  algorithm with symmetric key  $K$ . Let  $SHA2(M)$  denote the digest of the message  $M$  using  $SHA$ -256 hash algorithm. Herein the messages of the protocol implementation:

$$\begin{aligned} Ch(a) &= g^x || Pub_a \\ ChRe(b, Ch(a)) &= g^y || AES(SHA2(g^{xy}), RSA(Priv_b, g^x || g^y)) || Pub_b \\ Re(a, ChRe(b, Ch(a))) &= AES(SHA2(g^{yx}), RSA(Priv_a, g^x || g^y)) \end{aligned}$$

Note that all the exponentiations of  $g$  are done modulo  $p$ .

## 4.2 The case of wireless networks

We have saw that the protocol is secure only if underlying local area network respects **no hidden selective message destruction** property.

For example, Ethernet has property *no possible hidden selective message destruction*. Ethernet has a collision detection protocol so that the selective message destruction will be easily detected. This is achieved because each Ethernet device that sends a packet listens in the same time the carrier. If the listened packet is different from the expected packet a collision is detected. If the listened packet is the same as the expected packet, the device can assume that all devices in the LAN correctly received the packet. We can use the collision detection to have correct network security properties.

Wireless networks such as 802.11 do not have such collision detection. They just propose built-in non-secure collision avoidance. Thus, our protocol is not directly applicable to these networks. Nevertheless, an additional mechanism can be added to obtain collision detection (as for Ethernet). The main idea of the mechanism is based on the fact that packet transmission has some duration. A set of bits is transmitted on the communication channel, either one by one or group by group using orthogonal frequency-division multiplexing technology. Attacker and destination receive the same information in the same time. If an attacker wants to disturb only one message type and not all messages, he has to wait to get enough information before being able to decide whether the message shall be intercepted. The main idea is to transmit a flag denoted *protected* before the attacker can decide to disturb or not the message. The receiver receives this flag at the same time as the attacker.

There are two cases. If the message body is disturbed by an attacker (or by a normal collision), the receiver detects this disturbance and sends a message containing the flag *Coll*. If the receiver receives correctly the message, it sends a message containing the flag *NoColl*. Each of these flags (*protected*, *Coll* and *NoColl*) are placed in the packet before any information about source or destination.

When a device sends a message with flag *protected* set, it waits for responses. Three cases are possible:

1. only *NoColl* messages are received: the message has been sent correctly.

2. At least one *Coll* is received: there is an ongoing collision or attack.
3. No message is received: no device received the message.

Using this mechanism, our protocol can be extended to the case of 802.11 networks.

## Conclusion

We proposed a protocol that enables secure pairing of devices under realistic conditions. This protocol is user-friendly, as it does not require from the user to memorize or compare words or a string of characters. User has only to obey authoritative rules. Furthermore, this protocol is resistant to most of existing attacks like man-in-the-middle attacks, relay and replay attacks. Further work has to be done to increase the resistance to denial of service attacks.

*The authors would like to thank Nicolas Prigent for valuable remarks and support.*

## References

1. N. Prigent, J.P. Andreaux, C. Bidan, O. Heen, *Secure Long Term Communities In Ad Hoc Networks*, SASN workshop 2003.
2. Phonex Corps, *NerverWire14 User Guide*, [www.phonex.com](http://www.phonex.com).
3. Buffalo Tech, AOSS, [www.buffalotech.com/wireless/products/pdf/AOSS\\_DS.pdf](http://www.buffalotech.com/wireless/products/pdf/AOSS_DS.pdf).
4. Atheros, *JumpStart White paper*, 2004, [www.atheros.com/pt/whitepapers/atheros\\_JumpStart\\_for\\_wireless\\_whitepaper.pdf](http://www.atheros.com/pt/whitepapers/atheros_JumpStart_for_wireless_whitepaper.pdf).
5. Broadcom Corp, *SecureEasySetup*, 2005, [www.broadcom.com](http://www.broadcom.com).
6. G. Fleish, *Under the Hood with Broadcom SecureEasySetup*, 2005, [wifinetnews.com/archives/004685.html](http://wifinetnews.com/archives/004685.html).
7. M. Cagalj, J.P. Hubaux, *Key agreement over a radio link*, Technical report No. IC/2004/16, 2004.
8. J. Callas, et al., *RFC 2440, Open PGP Message Format*, IETF, 1998.
9. F. Stajano, R. Anderson, *The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks*, Lecture Notes in Computer Science, 1999.
10. W. Diffie, M. Hellman, *New Directions in cryptography*, IEEE Transactions on Information theory, 1976.
11. A. Menezes, et al., *An Efficient Protocol for Authenticated Key Agreement*, Technical report CORR 98-05, University of Waterloo, 1998.
12. A. Menezes, et al., *Handbook for Applied Cryptography*, CRC Press, 1996.
13. S. Blake-Wilson, et al., *Key Agreement Protocols and their Security Analysis*, In Proc. of 6th IMA International Conference on Cryptography and Coding, vol. 1355 of LNCS, pages 30-45. Springer, 1997
14. T. Aura, *Strategies against replay attacks*, In Proc. 10th IEEE Computer Security Foundations Workshop, 1997.