Contents

List of Figures

List of Tables

Chapter 1

Secure Device Pairing Protocols

The need to secure communications between personal devices is increasing nowadays, especially in the context of Internet of Things. Authentication between devices which have no prior common knowledge is a challenging problem. One solution consists in using a pre-authenticated auxiliary channel, human assisted or location limited, usually called out-of-band channel. A large number of device pairing protocols using an out-of-band channel were proposed. However most of these proposals lacks of proofs, and therefore may be vulnerable to some attacks. Additionally, current approaches are not sufficiently convenient for IoT applications where the devices are strictly constrained, and network bandwidth is too expensive.

In this part, we study in depth current secure device pairing protocols. We found that current approaches are not effective in term of computation and communication for Internet of Things applications. Therefore, we introduce a new key agreement protocol between two wireless devices. This protocol, only using two wireless messages and one out-of-band message, offers better communication costs than currently existing solutions, yet still ensuring a reasonable security. Security of our proposal is validated by estimation of attack success probability in a computational model.

The chapter begins with a short introduction to out-of-band channels, and existing device pairing schemes. Then, our novel device pairing will come after that. Finally, we are willing to introduce flaws we found in some current approaches.

1.1 Out-of-Band Channels

Securing wireless communication is establishing an initial trust relation between dissociated devices. Such a trust initialisation process is commonly called either *Secure*

Device Pairing, or *Secure Bootstrapping*, or *Secure First Connect.* Due to heterogeneity of devices and lacking of official standards, no existing security infrastructure or schemes could provide a universal solution for this task. Additionally, unfamiliar devices with no common trust cannot take advantage from traditional cryptographic protocols (i.e. authenticated key exchange protocols) when there does not exist any pre-shared secret, or authenticated public keys.

Trying to solve these problems, a great body of work proposes some forms of human involvement in secure pairing process. This human involvement is achieved by using an auxiliary channel between the devices that is both observable and controllable by human that manages the devices. This auxiliary channel received various names such as *out-of-band channel*, or *human-assisted channel* or *manual channel.* In this thesis, we adopt the general term out-of-band channel (OOB).

1.1.1 OOB Security Properties

One easily misunderstands concept of security properties of data or messages versus security properties related to physical channels because they sometimes overlap. Data security properties are usually ensured using cryptographic primitives such as symmetric or asymmetric encryption algorithms, signatures or hash functions. In meanwhile, a secure physical channel is able to provide not only data security properties without help of cryptographic mechanisms, but also physical security such as stall-free, listener-ready, non-forwarding, time and distance constraint guarantees and so on. In our chapter, we consider following security properties. The notation S and R represents for principals, m is a message, o is a channel, T is an interval of time:

- *Data Origin Authentication*[DOA]: Data origin authentication is also called *message authentication.* Let m be a message originally created by S, then any receiver of m is able to authenticate S as an original source of the message. It means that in a particular penetrator cannot modify m; thus message authentication includes message integrity. However, a penetrator may block or replay m.
- *Data Confidentiality*[DC]: If a sender determines that only a specified R can observe content of a message m, then no one including penetrators, yet except R is allowed to know the content of m.
- *Channel Occupancy*[CO] : If a specific receiver R uses a channel o to communicate with someone during interval of time T, then there is indeed a sender using o with R during T. A penetrator cannot manipulate o during T if o is not exclusively used by the penetrator.

- *Channel Origin Authentication*[COA] [1]: Only a specified sender S can use a channel o to send messages and this fact is known to determined receivers. Thus, a penetrator cannot impersonate S on o. However, a penetrator can suspend message transmission to know messages before they reach to their desired destination.
- *Channel Confidentiality*[CC] [1]: Only a specified receiver R can receive messages on a channel o, and this fact is known to determined senders. Thus, a penetrator cannot impersonate R to receive message on o. A penetrator cannot overhear messages sent on o.

Note that, channel occupancy allows participants to ensure distance and presence of their protocol partners. Straightforwardly, penetrators cannot use suspending attack on messages over such channels.

The definition of channel occupancy is an adaptation of locale occupancy property introduced in [2] and both the channel origin authentication and channel confidentiality properties were initially defined in[1]. The definitions above overlap: channel confidentiality implies data confidentiality and channel origin authentication implies data origin authentication.

1.1.2 OOB Classification

In this subsection, we classify out-of-band channels in two ways based on physical types of channels, and security properties that channels offer.

1.1.2.1 Physical Types of Channels

In this part, we classify existing approaches based on physical characteristics of channels into groups such as cable-based, audio-based, visual-based, tactile-based, motion-based, biometric-based, wireless-based, and combination. We refer this classification in [3].

Cable-based

Authors [4] proposed a resurrecting duckling model that maps relationship between devices. A master device, so-called ”Mother”, is a device which imprints ”duckling” slave devices. The slaves is either imprinted or imprint-able. Imprint-able state is the beginning state of a slave before it is chosen by a master. In meanwhile, the imprinted state is once a slave has got a secret from a master. The imprinted process actually bounds a slave to a master until the slave’s death. As a consequence, the slave remains faithful to its master and obeys no one else. Because the secret key needs to be transferred from a master to a slave, the authors suggest that it could be sent in plain text over a physical connection (such as cable). Complex and heavy cryptographic key exchange like DH scheme is not recommended. This thing, hence, is not convenient in practice. Nevertheless, this approach takes advantages of minimal requirement of human interaction in authentication phase.

Audio-based

Audio channels could be used as secure channels. The work [5–10] proposes ideas to encode cryptographic materials into nonsensical audio sentences. Then after transmitted from a speaker to a microphone, the sentences are reconstructed into the cryptographic materials at the target device. User takes responsibility for comparing results and deciding the pairing.

Audio-based schemes normally require some kinds of physical interfaces such as speaker, microphones. But, these interfaces are often suffered from denial of services or noise environments. For instance, ambient noise in crowded environments (e.g. in subway, in airport, or in bars) makes the authentication either weaker or difficult in speaker-to-speaker, as well as in others. Moreover, handicap users are not suitable for these schemes.

Recently, researchers have made improvement on both security levels and usability, thank to advanced speech engines and audio codec technologies.

Visual-based

Using image comparison to set up a secure channel between devices has appeared early in literature. Precisely, cryptographic materials are encoded into images, and ask users to compare them on two devices. Approaches chose this method such as [11–15]. Despite of the requirement of a high resolution screen on each device, these approaches stated that screens are easily found in current laptops, PDAs, smartphones, and etc.

Visual-based schemes also share the limitations of audio-based ones on hardware requirements. Furthermore, cameras are sometimes strictly prohibited in high security areas such as military zones or bank offices, and barcodes do not work in low light conditions.

Tactile-based

BEDA [16], proposed by Soriente et. al, presents ways to transmit a secret among devices using very basic interfaces like buttons. There are four BEDA variants: Button-to-Button, Display to Button, Short Vibration to Button, and Long Vibration to Button. The only difference of these variants is the way a first device transfers a secret to others.

To transmit the secret code, two approaches are suggested. In the first approach, both devices get the same secret via the use of single button. In the Button-Button approach, the user simultaneously presses and then releases buttons on both devices until the secret is acquired. In Display to Button approach, a device equipped with an output interface signals the user to press a button on the other device. Then, idle time between two pushing actions are is to calculate the secret.

Wireless-based

In order to establish a secure channel, some types of wireless channels such as infra-red [17], ultrasound [18], RFID [19], Bluetooth, and NFC could be used. Talking to Stranger and other its variants [17–20] are examples. However, a drawback of those schemes is that they are strongly suffered by denial of service attacks or passive eavesdropping attacks.

Pairing scheme using Bluetooth demands 4-digits pre-shared PIN code between two devices. Unfortunately, adversaries can guest and break the PIN code from long distance. As an alternative method, NFC, an extremely short communication, is concerned on solving limitations of Bluetooth and infra-red. In many scenarios, NFC combines with Bluetooth to offer quicker setup, and better security. Nevertheless, NFC does not provide any protection against eavesdropping attacks, as well as data corruption and data modification. In spite of that, it is hard to launch MITM attack in NFC authentication session. As this reason, NFC is still considered safety for current applications.

Biometric-based

A first work which tried to apply biometric data to establish a secure channel was found in Feeling-is-Believing [21] in which authors proposed a method to share secret key using authenticated biometric information.

In users' point of view, biometric-based schemes sound more secure and usable than others. But in practice, biometric processing is not sufficiently accurate and requires more calculation cost. To overcome this obstacle, thank to advanced technologies, the accuracy of recognition is significantly improved in many commercial devices such as modern smartphones and laptops. The only drawback of these such schemes is that biometric scanners must be equipped on both devices.

Accelerometer/Motion-based

Common uses of accelerometer include detecting and monitoring vibration, and detecting magnitude and direction. A first work using accelerometers for pairing devices was found in Smart-its-Friends scheme [22] in which two intended devices are held and shaken together simultaneously. By this way, the sensing information collected from accelerometers allows them to establish a common communication channel. Other variant approaches are [23–28].

A drawback of these approaches is that embedded accelerometers sometimes are not always easy to be deployed in big devices such as printers, projectors or laptops.

Combination

Claude Castelluccia and Pars Mutaf introduced *Shake Them Up* [29] to allow two constrained devices to share a secret in minimal requirements of both hardware and out-of-band channel. In this scheme, both involved devices are required to shake and twirl together in very proximity. During shaking process, both devices also exchange radio packets. When finishing the steps, they together get the same secret key. Attackers cannot interfere key exchange process because they cannot determine source of each radio packet. To limit the attackers' ability, the author exploited the source indistinguishability achieved by CDMA-based system, and sharking devices in close proximity.

Varshavsky et al. introduced AMIGO [30] investigated that radio signal fluctuations is too hard to predict at a specific location and time. With this result, the authors pointed that any attacker who is not physically close to the signal source would see a different pattern of signal strength.

1.1.2.2 Channel Security Properties

The types of OOB channels can be also presented via properties of channels. We refer the classification in the work [31], and adapt each type with the definitions of channel security properties. In addition to, we divide the public channel into sub-public channels.

- a *private channel* ensures all channel security properties;
- a *protected channel* ensures channel origin authentication and channel confidentiality but not channel occupancy;
- a *public channel* ensures channel origin authentication but not channel confidentiality;

Additionally we classify public channels into two sub-types:

- a *short-range(SR) public channel* is a public channel offering channel occupancy,
- a *long-range(LR) public channel* is a public channel not offering channel occupancy.

Table 1.1: Out-of-Band Channel Classification

OOB Channel Type	CO	COA	CC	Examples
Private	√	√	√	Cable
Protected	Ø	√	√	SMS, Encrypted email
Short-range Public	√	√	Ø	Button, Vibration, RFID, NFC...
Long-range Public	Ø	√	Ø	Screen to camera, Speaker to speaker, ...
Insecure	Ø	Ø	Ø	WIFI

Table 1.2: Threats on Out-of-Band Channels

Out of Band Channel	Attacker's Power			
Type	Overhear	Block	Suspend	Replay
Private	Ø	Ø	Ø	Ø
Protected	Ø	√	√	Ø
Short-range Public	√	√	Ø	Ø
Long-range Public	√	√	√	√

A private channel could be established for instance by connecting a cable between two devices. A protected channel could be set by using a tactile based technique like authenticated server SMS, authenticated emails. A short-range public channel can be offered by motion-based techniques or NFC. A long-range public channel can use a visual based or sound-based technique. Table 1.1 summaries a comparison of OOB channel and give some examples for each type of OOB channels.

1.1.3 OOB Penetrator Model

As described in previous subsections, the penetrator is prevented from performing actions on private OOB channels, yet he can still launch some malicious actions on public or protected OOB channels. Table 1.2 presents the penetrator power for each kind of OOB channels.

In the table 1.2, overhearing means that attackers are capable knowing OOB messages when they are being transmitted. Suspending means that attackers are able to suspend sending events. Especially, suspending attacker can completely know message's content before messages are transmitted on a public OOB channel. Blocking means that attackers can drop any message over an OOB channel. Replaying means attackers are capable replaying OOB messages.

The difference between types of OOB channels and penetrator power for each kind is summarised in table 1.3.

Table 1.3: Out-of-Band Channel Summarization

Out of Band Chanel			Attacker's Power			
Pairing Method	Interface	OOB Type	Overhear	Block	Suspend	Relay
Resurrecting Duckling Model	Cable	Private	Ø	Ø	Ø	Ø
Motion-based Model	Accelerometer	SR Public	√	√	Ø	Ø
BEDA Methods						
☐ Button-Button	Button	SR Public	√	√	Ø	Ø
☐ Display-Button	Display, Button	SR Public	√	√	Ø	Ø
☐ Vibration-Button	Accelerometer, Button	SR Public	√	√	Ø	Ø
Audio-based Methods						
☐ Audio Context Recognition	Speaker, Microphone	LR Public	√	√	√	√
☐ Speaker-Speacker	Speaker	LR Public	√	√	√	√
☐ Speaker-Microphone	Speaker, Microphone	LR Public	√	√	√	√
Visual-based Methods						
☐ Barcode-Camera	Barcode, Camera	LR Public	√	√	√	√
☐ Visual Comparison	Screen	LR Public	√	√	√	√
☐ Blinking Light	LED light	LR Public	√	√	√	√
Wireless-based Methods	Wireless Interface					
☐ GPRS/3G/4G		LR Public	√	√	√	√
☐ Infrared		SR Public	√	√	Ø	Ø
☐ NFC		SR Public	√	√	Ø	Ø

1.2 Overview on Secure Device Pairing Schemes

We refer the survey [31] and sum up existing device pairing proposals. Then, we point out their limitations. To begin with, we describe some notations used to picturize the protocol as follows:

- An arrow shows direction of each message. A solid line represents for an unsecured channel, while a dotted line represents for a secured (authenticated) channel.
- When Bob or Alice receives a message over a insecure channel, this message could be altered by attackers. An apostrophe appears on a letter, e.g. A' or M', it means that the received message may be different from the sent one.

A protocol utilises commitment schemes which commit on an arbitrary non-hidden message m together with a hidden k-bit string r. These schemes are formalised by three algorithms.

- **commit(M)** takes a message M and produces two strings: a commit value c and decommit value d.
- **open(c,d)** takes M or an error signal.

The notations used in our report are summarised in Table1.4

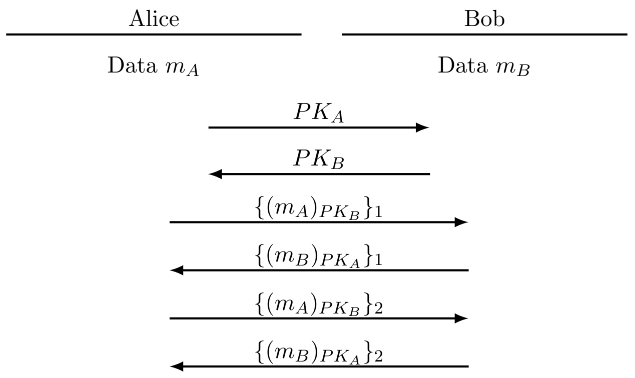
Table 1.4: The Notations

Symbol	**Meaning**
Aice(A) and Bob(B)	Communicating entities
ID_X	Device X identifier
r_x	Random number generated by entity X
K_X	Key generated by entity X
PK_X	Public Key of entity X
m	Data
$MAC_K(m)$	Message authentication code of message M using shared key K
$h_K(K)$	Universal hash function of message M using the key K
h(m)	One-way hash function of message M
,	Concatenation of different parts of a message
$\oplus$	Bitwise "XOR" operation
g^x	Diffie-Hellman public key entity X
g^{xy}	Diffie-Hellman shared key between X and Y
$(c, d) \leftarrow commit(m)$	commitment algorithm takes m to produce commit value c and decommit d
$m \leftarrow open(c, d)$	Open commitment algorithm take c, d and produces a message m or error signal

1.2.0.1 Interlock Protocol

Proposed by Rivest and Shamir in 1984, Interlock protocol exploits user's knowledge of communication pattern or voice recognition to eliminate eavesdropping attacks. In this scheme, two parties use their initial knowledge to mutually authenticate their public keys without any assistance of a third party. This protocol works as follows:

1. Alice and Bob exchange their public keys via a sound channel.
2. Alice produces $(m_A)_{PK_B}$ then sends the first half of the result to Bob.
3. Bob produces $(m_B)_{PK_A}$ then sends the first half of the result to to Alice.
4. Alice sends to Bob the second half of $(m_A)_{PK_B}$.
5. Bob sends to Alice the second half of $(m_B)_{PK_A}$.
6. Both decrypt the combination of two halves with their own private key.

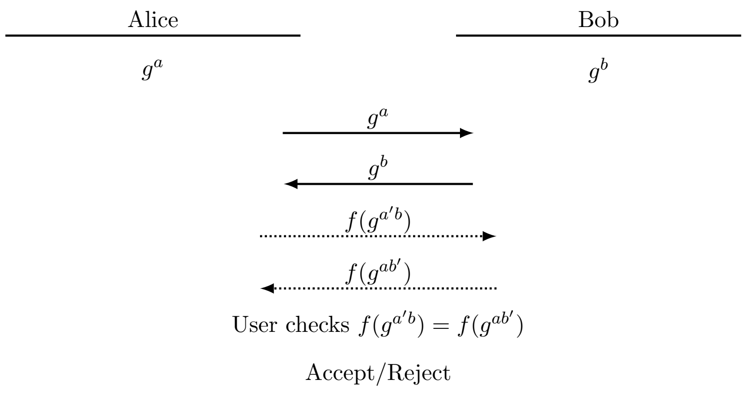


The strength of this protocol basically lies on encryption algorithms with which the attacker cannot decrypt the received halves of encrypted messages. In case of that the attacker intentionally send some new messages to destroy the communication, he will be revealed.

1.2.0.2 Maher Manual Authentication

In 1993, Maher got his patent for several pairing methods [32] allowing a user to share secret DH key manually. In one method, he applied a compression function to interpret DH key g^{ab} into 4-digit number. The number than is showed on each device screen. A user gets easy to compare two numbers on both devices. The protocol simply happens as follows:

1. Alice and Bob generate DH value g^a and g^b respectively.
2. Alice and Bob exchange their values.
3. Both calculate $f(g^{ab})$ to 4-digit hex number, and show the result on their device's screen.
4. The user decides accept or reject by pushing a button.



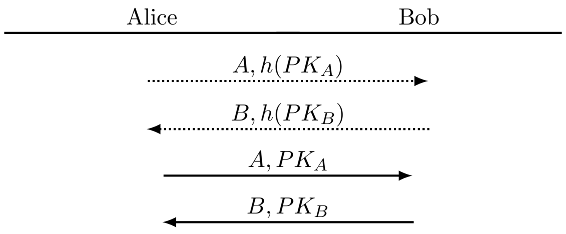
The protocol requires at least 80 bits over an out-of-band channel to get enough secure. At a result, this length of bit string is over-weighted on any out-of-band channel.

1.2.0.3 Talking to Strangers

Balfanz et al. proposed a new scheme in [33] using audio, or infrared channels to transmit a hash of public keys which is considered as a fingerprinting or digest value. The protocol works as follows:

1. Alice and Bob initially exchange their identifications and a hash of their pubic keys over an out-of-band channel.

2. Both sides exchange their ID and their public key on insecure channel.



Security of this protocol mainly depends on security properties of out-of-band channels and the hash function. To prevent MITM attack, this protocol needs at least 80 bits information over each direction of OOB channel. Otherwise, the attacker can find out the pair of public keys PK'_A/PK'_B in which $h(PK_A) = h(PK'_A)$ and $h(PK_B) = h(PK'_B)$ to launches a MITM attack.

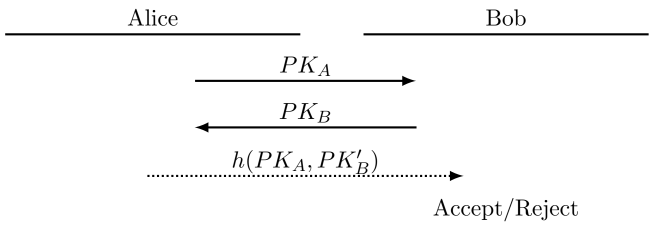
1.2.0.4 Visual authentication based on Integrity Checking

Saxena et al in [15] proposed a new pairing protocol, namely Visual authentication based on Integrity Checking(VIC) which works as follows:

1. Alice and Bob initially exchange their pubic keys.

2. Alice hashes both keys, and sends the hashing value to Bob.

3. After verification the Alice's hashsing value, Bob informs Accept or Reject back to Alice.



Security of VIC heavily depends on strength of the hash function and requires at least 80 bits on each OOB channel to prevent hash collision.

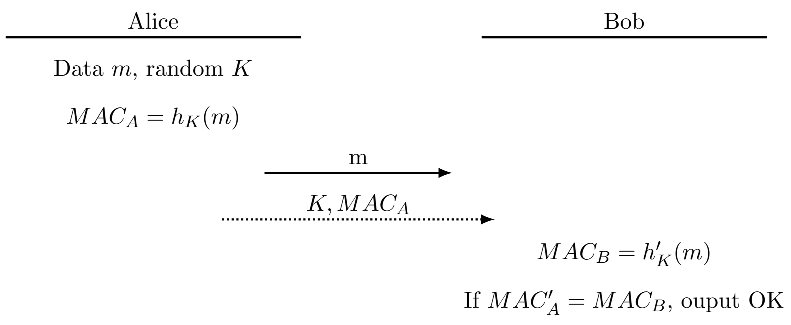
1.2.0.5 Manual Authentication(MANA) Protocols

Due to the long length of OOB messages in [15], and [33], some studies tried to truncate this length into 16 or 32 bits (4 or 8 hexadecimal digits, respectively), this could lead to security weakness. Adversaries probably recover the messages from the hash-codes. To overcome this problem, Christian Gehramann and Chirs J.Michell [34] proposed three type of manual authentication protocols: MANA I for Output-Input, MANA II for Output-Output, and MANA III for Input-Input. These protocols remarkably reduce bandwidth of OOB channel to k bits (16-20 bits) in each way while a MITM attack success probability is still hold at 2^{-k}.

MANA I protocol

The authors presented a first example scheme [34], which is designed for situation in which one device has a keyboard, the other has a display. Although the scheme uses keyed check-function with short check-value (16 to 20 bits), it is proved to provide sufficient secure.

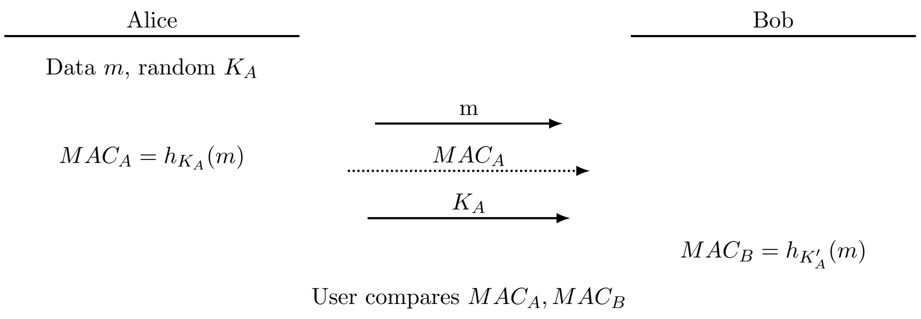
1. *Alice* sends a message m to *Bob* over an insecure channel.
2. *Alice* generates a random key K(16-20) bits. *Alice* also generates $h_K(m)$, then output to its display.
3. User enters $h_K(m)$ and K to *Bob*.
4. *Bob* uses K and recomputes $h_K(m)$, then compares the value that the user has entered.
5. The user copies success or failure.



MANA II protocol

MANA II [35] is a variant of MANA I. In this scheme, both devices are equipped with displays and keyboards. The protocol works as follows:

1. *Alice* sends a message m to *Bob* over an insecure channel.
2. *Alice* generates a random key K_A(16-20) bits. *Alice* also generates $h_{K_A}(m)$, then output it to the user over a secured channel.
3. *Alice* sends key K_A to *Bob* over the insecure channel.
4. *Bob* uses K_A and recomputes $h_{K_A}(m)$, then sends $h_{K_A}(m)$ to the user.
5. The user compares two values from both devices. Then if results are matched, the user indicates success to both devices over a secured channel.

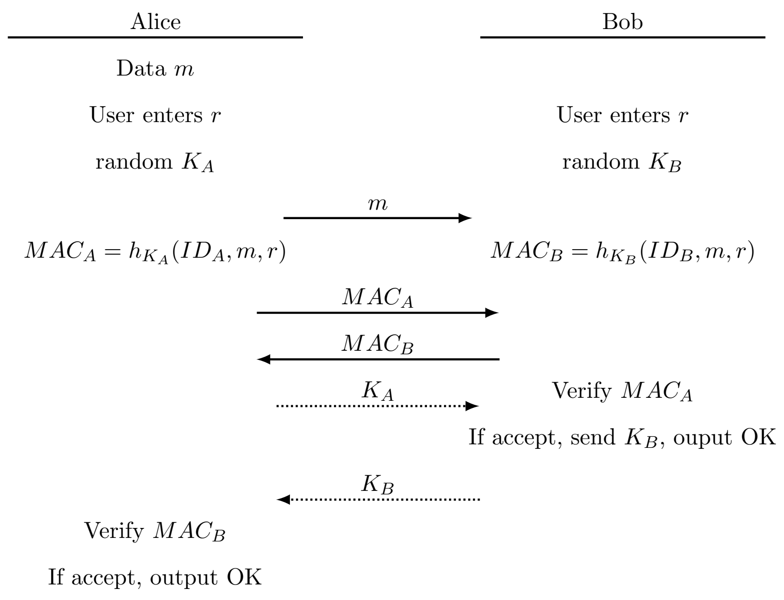


MANA III protocol

MANA III protocol [34] is designed for a situation in which both devices have keyboards. Both devices are assumed on agreement on a public data M. Here, $m_K(M)$ denotes a MAC value computed using a key K and a data string M. I_A and I_B are identifications of *Alice* and *Bob*, respectively. The scheme operates as follows.

1. *Alice* sends a message m to *Bob* over an insecure channel.
2. A user generates a short random bit-string (16-20) bits r and enters it to 2 devices.
3. *Alice* generates a key K_A, computes $MAC_A = h_{K_A}(ID_A, m, r)$, then sends MAC_A to *Bob* over a wireless link.
4. *Bob* generates a key K_B, computes $MAC_B = h_{K_B}(ID_B, m, r)$, then sends MAC_B to *Alice* over the wireless link.

5. When *Alice* receives MAC_B, then sends K_A to *Bob* over a secured channel.
6. When *Bob* receives MAC_A, then sends K_B to *Alice* over the secured channel.
7. *Alice* recomputes MAC_B and verifies its stored value of m, the expected identifier I_B, and random value r.
8. *Bob* recomputes MAC_A and verifies its stored value of m, the expected identifier I_A, and random r.
9. If(and only if) both devices indicate success, the user indicates success two both devices.



Analysis of MANA Protocols

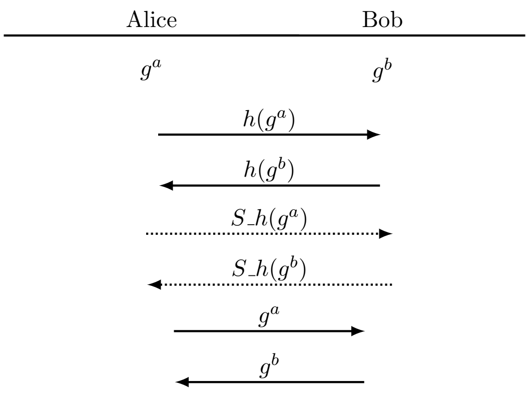
An out-of-band channel used in MANA I and II could be a public or protected channel, while MANA III strongly requires a private channel as the key R must be confident. If leaking R, MANA III protocol is definitely a victim of MITM attack. Furthermore, success probability of attacks in MANA protocol is 2^{-k} where k is length of check-value messages in MANA I and MANA II, and is the length of R in MANA III. In term of optimisation, in MANA I and II, the user must compares check-values and the key K. However, in term of security, MANA I may provide a stronger security level than others.

1.2.0.6 Ephemeral Pairing Protocols

Hoepman AKA Protocol

Jaap-Henk Hoepman in [36] proposed his protocols by exploited out-of-band channel properties. In his protocol, parties securely share their DH public keys in two phases. First, long hashes of both sides' public keys are exchanged over insecure channels. Second, short hashes are sent over out-of-band channels. The detail protocol is illustrated as follows.

1. Alice and Bob generate DH-value g^a and g^b respectively.
2. Both sides exchange a long hashing value of g^a and g^b over an insecure channel.
3. After the reception of the long hashing value, both sides calculate a short hashing value of g^a and g^b, exchange them over an out-of-band channel.
4. At the end, both sides reveal their value g^a and g^b to each other in the insecure channel.

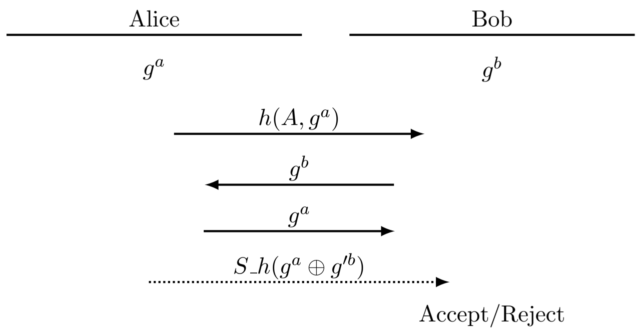


Improved Ephemeral Pairing

Nguyen and Roscoe [37] proposed an improved version of Hoepman protocol by cutting off one message on OOB channel. The protocol works as follows:

1. Alice and Bob generate DH-value g^a and g^b respectively.
2. Alice sends $h(A, g^a)$ to Bob.
3. Both sides exchange g^a and g^b to each other.

4. At the end, Alice calculates a short hash of $g^a \oplus g'^b$, then sends it to Bob over an out-of-band channel.



Analysis of Ephemeral Protocols

Security of Hoepman protocol heavily depends on freshness of DH public keys in each protocol session. Otherwise, an attacker is able to discover a matching key with the same short hash output, then successfully launches a MITM attack. Hoepman also provided a proof of his protocol in the Bellare-Pointcheval- Rogaway model. The improvement scheme of Nguyen and Roscoe remains the same requirement.

1.2.0.7 Wong-Stajano Multichanel Security Protocols

Wong and Stajano [38] proposed new mutual authentication and key agreement protocols over bidirectional and unidirectional public out-of-band channels. Their protocols exploit a short authenticated string over visual channels which provide data origin authenticity.

Protocol with Bidirectional Channel

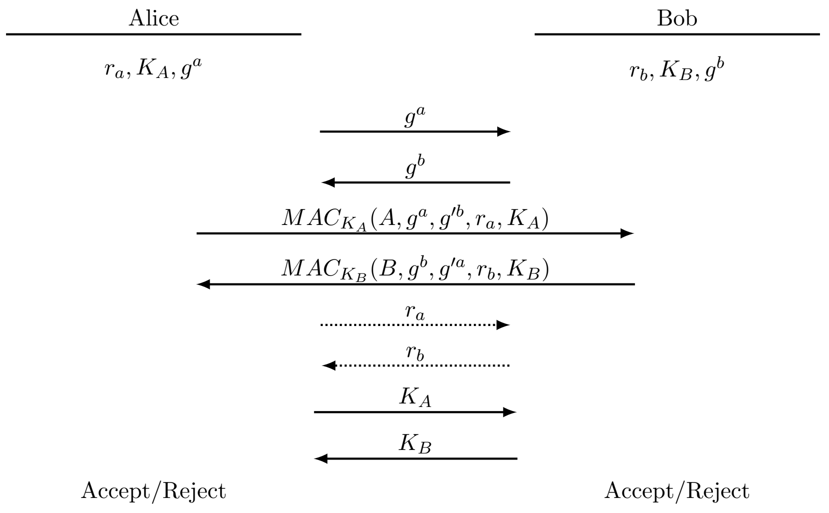
Wong and Stajano introduced a new variant of MANA III protocol which works as follows:

1. Alice generates a random number r_a, a key K_A, and DH-value g^a.
2. Bob generates a random number r_b, a key K_B, and DH-value g^b.
3. Both sides exchange g^a and g^b to each other.
4. Alice calculates $MAC_{K_A}(A, g^a, g'^b, r_a, K_A)$, then sends it to Bob.

5. Bob calculates $MAC_{K_B}(B, g^b, g'^a, r_b, K_B)$, then sends it to Alice.

6. Both sides exchange their r_a and r_b to each other over an out-of-band channel.

7. At the end, Bob sides exchange their K_A and K_B to each other.



In term of usability, the above protocol spends 6-move insecure communication, and long hash of DH public keys that put a heavy pressure on constrained devices. Realising this limitation, the authors improved their first work by an improved version over the bidirectional public out-of-band channel. The new protocol works as follows:

1. Alice generates a random number r_a, a key K_A, and DH-value g^a.

2. Bob generates a random number r_b, a key K_B, and DH-value g^b.

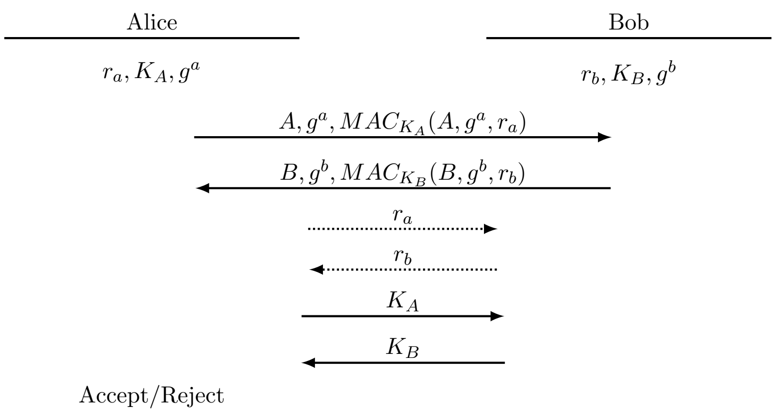
3. Alice calculates $MAC_{K_A}(A, g^a, g'^b, r_a, K_A)$, then sends it with Alice's identification A and g^a to Bob.

4. Bob calculates $MAC_{K_B}(B, g^b, g'^a, r_b, K_B)$, then sends it with Bob's identification B and g^b to Alice.

5. Both sides exchange their r_a and r_b to each other over an out-of-band channel.

6. At the end, Bob sides exchange their K_A and K_B to each other.

7. Alice informs to Bob Accept or Reject the session.

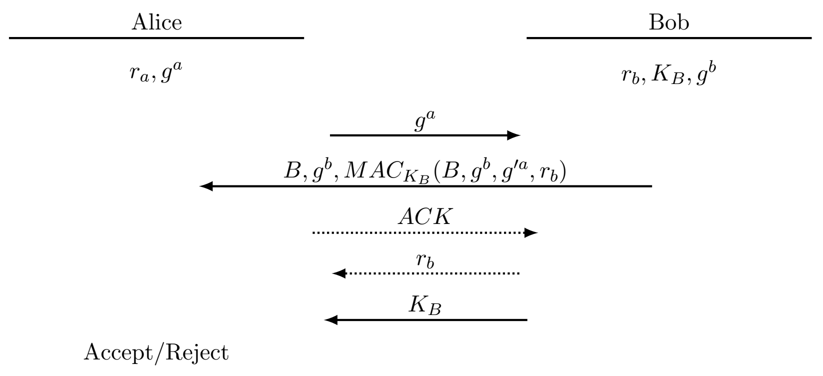


The improved protocol combines the first 4 messages of MANA III variant to 2 messages, and uses the MAC function, instead of a general hash function.

Protocol with Unidirectional Channel

In the same paper, Wong and Stajano also proposed a new protocol with unidirectional out-of-band channel. This version only spend 3 move on the wireless channel rather than 4-move in bidirectional version. The protocol works as follows:

1. Alice generates a random number r_a and DH-value g^a.
2. Bob generates a random number r_b, a key K_B, and DH-value g^b.
3. Alice sends g^a to Bob.
4. Bob calculates $MAC_{K_B}(B, g^b, g'^a, r_b, K_B)$, then sends it with Bob's identification B and g^b to Alice.
5. Alice informs to Bob that she has already received his message over an out-of-band channel.
6. Bob sends r_b to Alice over the out-of-band channel.
7. Bob sends K_B to Alice.
8. Alice informs to Bob Accept or Reject the session.

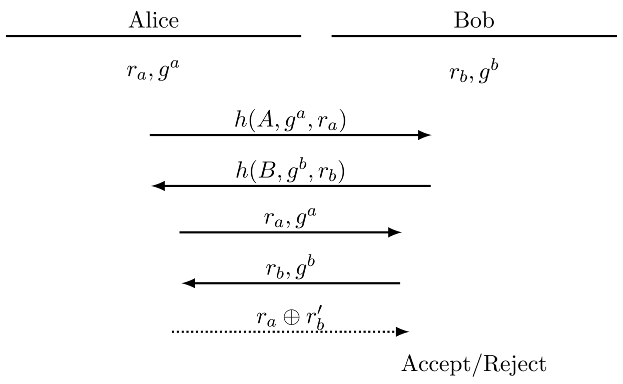


In term of efficiency, this unidirectional protocol only spends 5 messages including 2 out-of-band messages, that decreases both computation and communication cost compared to costs of two previous protocols.

Improved Wong-Stajano Key Agreement Protocol

Nguyen and Roscoe [37] proposed an variant of Wong-Stajano protocol. The new scheme cuts off long keys, and replaces two different authenticated string r_a and r_b by a single value $r_a \oplus r'_b$. The protocol works as follows:

1. Alice generates a random number r_a and DH-value g^a.
2. Bob generates a random number r_b, and DH-value g^b.
3. Alice calculates $h(A, g^a, r_a)$, and sends it to Bob.
4. Bob calculates $h(B, g^b, r_b)$, then sends it to Alice.
5. Alice sends r_a and g^a to Bob.
6. Bob sends r_b and g^b to Alice.
7. Alice calculates $r_a \oplus r'_b$, then pushes it on an out-of-band channel to Bob.
8. Bob informs to Alice Accept or Reject the session.



Analysis of Wong-Stajano Protocols

Wong-Stajano protocols were designed against either passive or active attacks. Passive attacks are resisted by DH components, while active attacks are resisted by short hash values over OOB channel. However, our work has successfully exploited the protocols. The counterexample will be presented in the following section.

1.2.0.8 Short Authenticated String-Based Authentication Key Agreement Protocols

MANA-based protocol family usually requires a strong assumption on a channel on which adversaries are not able to delay or relay any OOB message. To ease this strict condition, Serge Vaudenay introduced a protocol [39] based on Short Authentication String (SAS) in 2005. This scheme uses k-bit on OOB channel, and still preserves 2^{-k} attack success probability.

4-Move SAS-based Mutual-Authentication

Vaudenay presented his first authentication protocol based on a short authenticated string over an OOB channel [39]. The protocol is illustrated as follows.

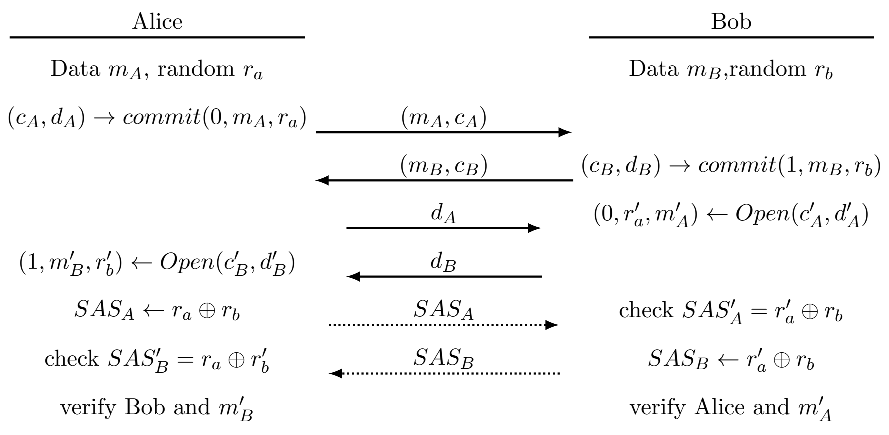
1. Alice types a message m_A, then picks a random value r_a. Bob types a message m_B, then picks a random value r_b
2. Alice computes $(c_A, d_A) \leftarrow commit(0, m_A, r_a)$, and sends m_A, c_A to Bob.
3. Bob computes $(c_B, d_B) \leftarrow commit(1, m_B, r_b)$ sends m_B, c_B to Alice.
4. Alice sends d_A to Bob. Then Bob computes $(0, r'_a, m'_A) \leftarrow Open(c'_A, d'_A)$.

5. Bob sends d_B to Alice. Then Alice computes $(1, r'_b, m'_B) \leftarrow Open(c'_B, d'_B)$.

6. Alice computes $SAS_A \leftarrow r_a \oplus r'_b$, and sends SAS to Bob over an out-of-band channel.

7. Bob computes $SAS_B = r'_a \oplus r_b$, then sends SAS_B back to Alice over his out-of-band channel.

8. Both sides compare their calculated SAS value to received one from the other.


Alice
Bob
Data m_A, random r_a
Data m_B,random r_b
$(c_A, d_A) \to commit(0, m_A, r_a)$
(m_A, c_A)
(m_B, c_B)
$(c_B, d_B) \to commit(1, m_B, r_b)$
d_A
$(0, r'_a, m'_A) \leftarrow Open(c'_A, d'_A)$
$(1, m'_B, r'_b) \leftarrow Open(c'_B, d'_B)$
d_B
$SAS_A \leftarrow r_a \oplus r_b$
SAS_A
check $SAS'_A = r'_a \oplus r_b$
check $SAS'_B = r_a \oplus r'_b$
SAS_B
$SAS_B \leftarrow r'_a \oplus r_b$
verify Bob and m'_B
verify Alice and m'_A

3-Move SAS-based Mutual-Authentication

Sylvain Pasini and Serge Vaudenay [40] attempted to decrease interaction cost of the protocol [39] down to 3 moves by advantage of a random oracle model. The protocol runs as follows.

1. Alice types a message m, then picks a random value r_a. Bob types a message m, then picks a random value r_b

2. Alice computes $(c_A, d_A) \leftarrow commit(m, r_a)$, and sends c to Bob.

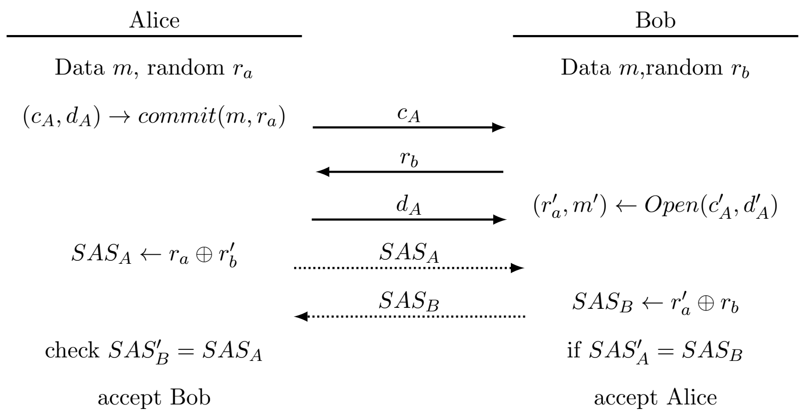
3. Bob sends r_b to Alice.

4. Alice sends d_A to Bob. Then Bob computes $(r'_a, m') \leftarrow Open(c'_A, d'_A)$.

5. Alice computes $SAS_A \leftarrow r_a \oplus r'_b$, and sends SAS to Bob over an out-of-band channel.

6. Bob computes $SAS_B = r'_a \oplus r_b$, then sends SAS_B back to Alice over his out-of-band channel.

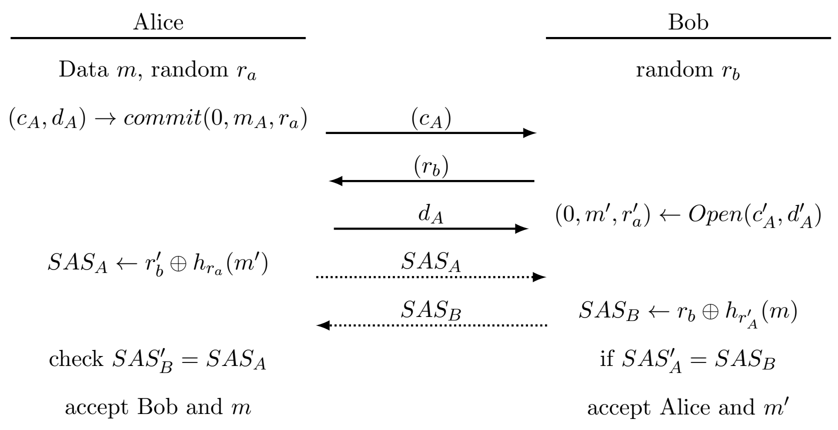
7. Bob verifies SAS and Alice. Alice compares both values of SAS and verifies Bob.


Alice
Bob
Data m, random r_a
Data m,random r_b
$(c_A, d_A) \rightarrow commit(m, r_a)$
c_A
r_b
d_A
$(r'_a, m') \leftarrow Open(c'_A, d'_A)$
$SAS_A \leftarrow r_a \oplus r'_b$
SAS_A
SAS_B
$SAS_B \leftarrow r'_a \oplus r_b$
check $SAS'_B = SAS_A$
if $SAS'_A = SAS_B$
accept Bob
accept Alice

3-Move SAS-based Cross Authentication

New SAS-based Cross Authentication based on the previous 3-move mutual authentication protocol improves a number of exchanged messages and uses a strongly universal hash function family. The protocol is presented as follow.

1. Alice types message m, then picks a random value r_a. Bob picks a random value r_b
2. Alice computes $(c_A, d_A) \leftarrow commit(0, m, r_a)$, and sends (c_A) to Bob.
3. Bob sends (r_b) to Alice.
4. Alice sends d_A to Bob. Bob computes $(0, m', r'_a) \leftarrow Open(c'_A, d'_A)$.
5. Alice computes $SAS_A \leftarrow r'_b \oplus h_{r_a}(m)$, and send SAS_A to Bob through out-of-band channel.
6. Bob computes $SAS_B \leftarrow r_b \oplus h_{r'_a}(m')$ and send SAS_B to Alice through out-of-band channel.
7. Alice and Bob check received SAS with their own SAS. Alice verifies Bob and m. And, Bob verifies Alice and m.

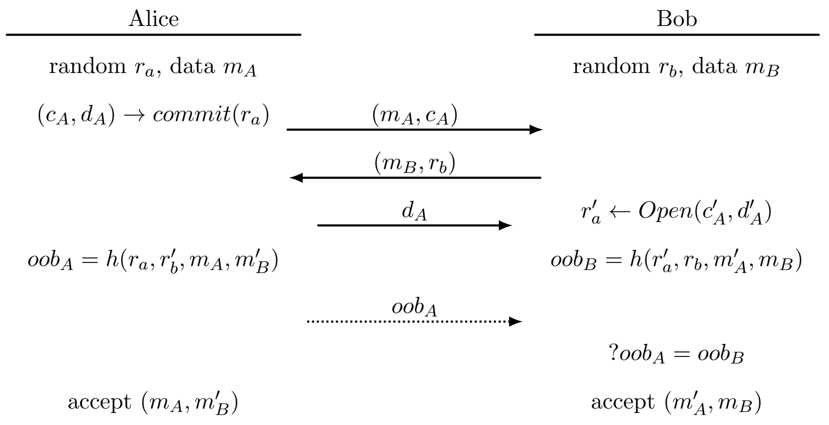

Alice
Bob
Data m, random r_a
random r_b
$(c_A, d_A) \rightarrow commit(0, m_A, r_a)$
(c_A)
(r_b)
d_A
$(0, m', r'_a) \leftarrow Open(c'_A, d'_A)$
$SAS_A \leftarrow r'_b \oplus h_{r_a}(m')$
SAS_A
SAS_B
$SAS_B \leftarrow r_b \oplus h_{r'_A}(m)$
check $SAS'_B = SAS_A$
if $SAS'_A = SAS_B$
accept Bob and m
accept Alice and m'

MANA IV Protocol

Presented a new method based on Vaudenay protocol [39], Sven Laur and Kaisa Nyberg [41] claimed that their proposal was more general, and had weaker security assumptions than one in [40]. Three round cross-authentication protocol Mana IV with l-bit OOB messages is presented as follows.

1. Alice computes $(c_A, d_A) \leftarrow commit(r_a)$ for random r_a, and sends (m_A, c_A) to Bob.
2. Bob chooses random r_b, and sends (m_B, r_b) to Alice.
3. Alice sends d_A Bob.
4. Bob computes $r'_a \leftarrow Open(c_A, d_A)$ and halts if $r'_a = NULL$. Both parties compute a test value $oob = h(r'_a, r_b, m_A, m_B)$ from received messages.
5. Alice sends oob_a to Bob over a secure channel.
6. Both parties accept (m_A, m_B) iff the local l-bit test value oob_a and oob_b coincide.

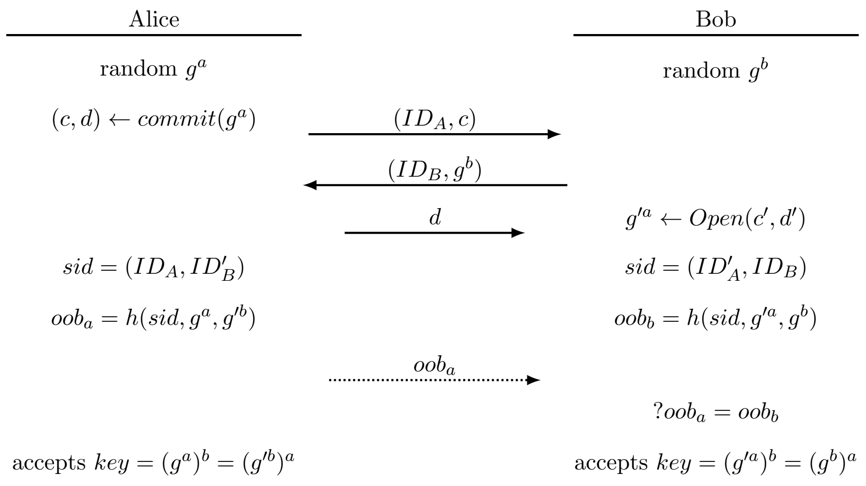
Specification: h is a keyed hash function with sub-keys K_A, K_B.



Manually authenticated MA-DH

When revising MANA IV protocol, Laur and Kyberg [41] indicated that the protocol was not computational efficiency. For this reason, they proposed a new modified protocol, namely Manually Authenticated Diffie-Hellman (MA-DH). The protocol runs as follows.

1. Alice computes $(c, d) \leftarrow commit(g^a)$, and sends (ID_A, c) to Bob.
2. Bob computes g^b for a random b, and sends (ID_B, g^b) to Alice.
3. Alice sends d to Bob.
4. Bob computes $K'_A \leftarrow Open(c', d')$ and halts if $g'^a = NULL$. Both parties compute $sid = (ID_A, ID_B)$ and $oob = h(sid, g^a, g^b)$.
5. Both parties accept $key = (g^a)^b = (g^b)^a$ iff the l-bit test values oob_a and oob_b coincide.


Alice
Bob
random g^a
random g^b
$(c, d) \leftarrow commit(g^a)$
(ID_A, c)
(ID_B, g^b)
d
$g'^a \leftarrow Open(c', d')$
$sid = (ID_A, ID'_B)$
$sid = (ID'_A, ID_B)$
$oob_a = h(sid, g^a, g'^b)$
$oob_b = h(sid, g'^a, g^b)$
oob_a
$?oob_a = oob_b$
accepts $key = (g^a)^b = (g'^b)^a$
accepts $key = (g'^a)^b = (g^b)^a$

Analysis of SAS-Based Protocols

Vaudenay and Pasini used Bellare-Rogaway model to prove their protocols. They also claimed that attack success probability is 2^{-k} for k-bit SAS, and $Q_A Q_B 2^{-k}$ where Q_A and Q_B are a number of instance of Alice and Bob in network. For instance, k is 50-bits in multi-party network, and 15-bits in 2-party network.

However, Laur and Nyberg in [41] did not agree with Vaudenay results because their proofs are not sufficient, and Random Oracle(RO) model and Common Reference String (CRS) model are not suitable for ad hoc networks.

Laur and Nyberg showed in their model that attack success probability against MANA IV and DA-DH is 2^{-k} where k is length of SAS value.

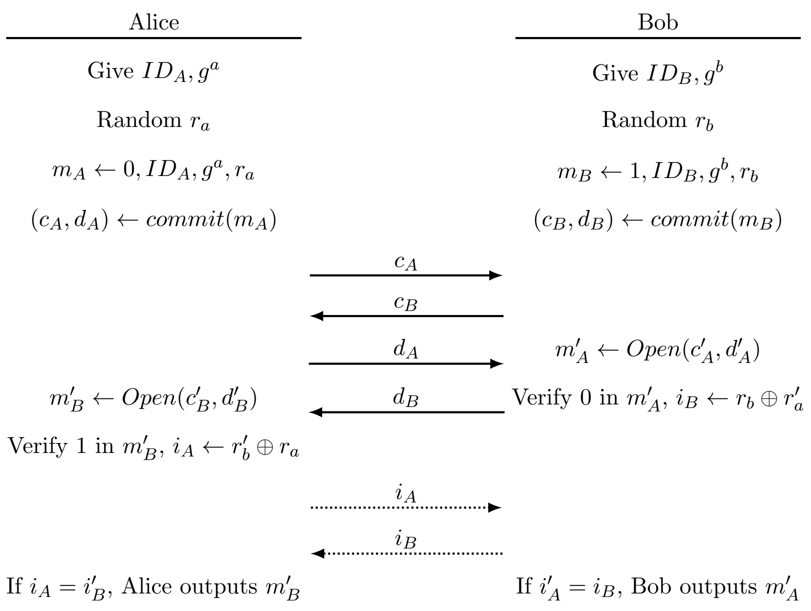
1.2.0.9 Cagalj-Capkun-Huxbaux Protocols

Cagalj, Capkun and Huxbaux [42] proposed three various protocols which are based on original DH protocol, and are assisted by human operations. The first protocol is based on visual comparison of short strings (DH-SC), the second one on distance bounding(DH-DB), and the third one on integrity codes(DH-IC). Each of them is presented below.

Diffie-Hellman key agreement protocol with String Comparison

1. Alice and Bob selects secret exponents a and b, and choose random number r_a and r_b respectively.
2. Alice and Bob calculate DH public parameters g^a and g^b.
3. Alice computes $m_A \leftarrow 0, ID_A, g^a, r_a$. Bob computes $m_B \leftarrow 1, ID_B, g^b, r_b$.
4. Alice computes $(c_A, d_A) \leftarrow \mathit{commit}(m_A)$ and sends (c_A) to Bob.
5. Bob computes $(c_B, d_B) \leftarrow \mathit{commit}(m_B)$, and sends (c_B) to Alice.
6. Alice sends d_A to Bob. Bob computes $m'_A \leftarrow \mathit{Open}(c'_A, d'_A)$ and verifies that 0 appears at beginning of m'_A. If verification is successful, Bob sends d_B to Alice.
7. Alice computes $m'_B \leftarrow \mathit{Open}(c'_B, d'_B)$ and verifies that 1 appears at beginning of m'_B
8. If the verification of Alice is successful, both parties go to the next stage.
9. Alice computes $i_A \leftarrow r'_b \oplus r_a$, Bob computes $i_B \leftarrow r_b \oplus r'_a$.
10. Bob sides exchange their short values over an out-of-band channel. If the received value equals the computed value, each side informs a successful signal.

Note that, 0 and 1 are public values used to prevent reflection attack. The identification ID_A and ID_B are human readable, e.g. email addresses or names. The authors used screens or human-voice as an out-of-band channel to securely transmit authenticated values.


Alice
Bob
Give ID_A, g^a
Give ID_B, g^b
Random r_a
Random r_b
$m_A \leftarrow 0, ID_A, g^a, r_a$
$m_B \leftarrow 1, ID_B, g^b, r_b$
$(c_A, d_A) \leftarrow commit(m_A)$
$(c_B, d_B) \leftarrow commit(m_B)$
c_A
c_B
d_A
$m'_A \leftarrow Open(c'_A, d'_A)$
$m'_B \leftarrow Open(c'_B, d'_B)$
d_B
Verify 0 in m'_A, $i_B \leftarrow r_b \oplus r'_a$
Verify 1 in m'_B, $i_A \leftarrow r'_b \oplus r_a$
i_A
i_B
If $i_A = i'_B$, Alice outputs m'_B
If $i'_A = i_B$, Bob outputs m'_A

Diffie-Hellman key agreement protocol with Distance Bounding

Distance bounding protocol allows both devices to verify the distance between them. Hence, this protocol can act as a secure channel.

Protocol DH-DB is mainly built on the DH-SC. Upon reception of commitment d_A, d_B, both devices execute distance bounding protocol to exchange r_a, r_b, i_A, i_B. According to time-flight estimation, each device can extract upper bound distance of the other devices.

Finally, both devices exchange y_A, and y_B to open the commitment x_A, x_B respectively. At the last step, each device check if i'_A, i'_B and i_A, i_B equal or not. Note that, this last step is done by device itself, whereas in DH-SC the comparison is performed by the users.

Alice		Bob
Give ID_A, g^a		Give ID_B, g^b
Random r_a, r_a		Random r_b, r_b
$m_A \leftarrow 0, ID_A, g^a, r_a$		$m_B \leftarrow 1, ID_B, g^b, r_b$
$(c_A, d_A) \leftarrow \mathit{commit}(m_A)$		$(c_B, d_B) \leftarrow \mathit{commit}(m_B)$
$(x_A, y_A) \leftarrow \mathit{commit}(0, r_a)$		$(x_B, y_B) \leftarrow \mathit{commit}(1, r_b)$
	$\xrightarrow{c_A, x_A}$	
	$\xleftarrow{c_B, x_B}$	
	$\xrightarrow{d_A}$	$m'_A \leftarrow \mathit{Open}_{(}c'_A, d'_A)$
$m'_B \leftarrow \mathit{Open}_{(}c'_B, d'_B)$	$\xleftarrow{d_B}$	Verify 0 in m'_A
		$i_B \leftarrow r_b \oplus r'_a$
Verify 1 in m'_B		
$i_A \leftarrow r'_b \oplus r_a$		
	-DB Phase-	
r_a bits $= r_{a1}, .., r_{ak}$		r_b bits $= r_{b1}, .., r_{bk}$
i_A bits $= i_{A1}, .., i_{Ak}$		i_B bits $= i_{B1}, .., i_{Bk}$
$\alpha_1 \leftarrow r_{a1} \oplus i_{A1}$	$\overset{\alpha_1}{\dashrightarrow}$	
	$\overset{\beta_1}{\dashleftarrow}$	$\beta_1 \leftarrow r_{b1} \oplus i_{B1} \oplus \alpha'_1$
	...	
$\alpha_i \leftarrow r_{ai} \oplus i_{Ai} \oplus \beta'_{i-1}$	$\overset{\alpha_i}{\dashrightarrow}$	
	$\overset{\beta_i}{\dashleftarrow}$	$\beta_i \leftarrow r_{bi} \oplus i_{Bi} \oplus \alpha'_i$
	...	
$\alpha_k \leftarrow r_{ak} \oplus i_{Ak} \oplus \beta'_{k-1}$	$\overset{\alpha_k}{\dashrightarrow}$	
	$\overset{\beta_k}{\dashleftarrow}$	$\beta_k \leftarrow r_{Bk} \oplus i_{Bk} \oplus \alpha'_k$
	-End of DB Phase-	
	$\xrightarrow{y_A}$	$0, r'_a \leftarrow \mathit{Open}(x'_A, y'_A)$
$1, r'_b \leftarrow \mathit{Open}_{(}x'_B, y'_B)$	$\xleftarrow{y_B}$	
$i'_{Bi} \leftarrow \alpha_i \oplus \beta'_i \oplus r'_{bi}$ (i=1,..,k)		$i'_{A1} \leftarrow \alpha_i \oplus r'_{a1}$
		$i'_{Ai} \leftarrow \alpha'_i \oplus \beta_{i-1} \oplus r'_{ai}$ (i=2,..,k)
If $i_A = i'_B$, Alice outputs m'_B		If $i'_A = i_B$, Bob outputs m'_A

Diffie-Hellman key agreement protocol with Integrity Codes

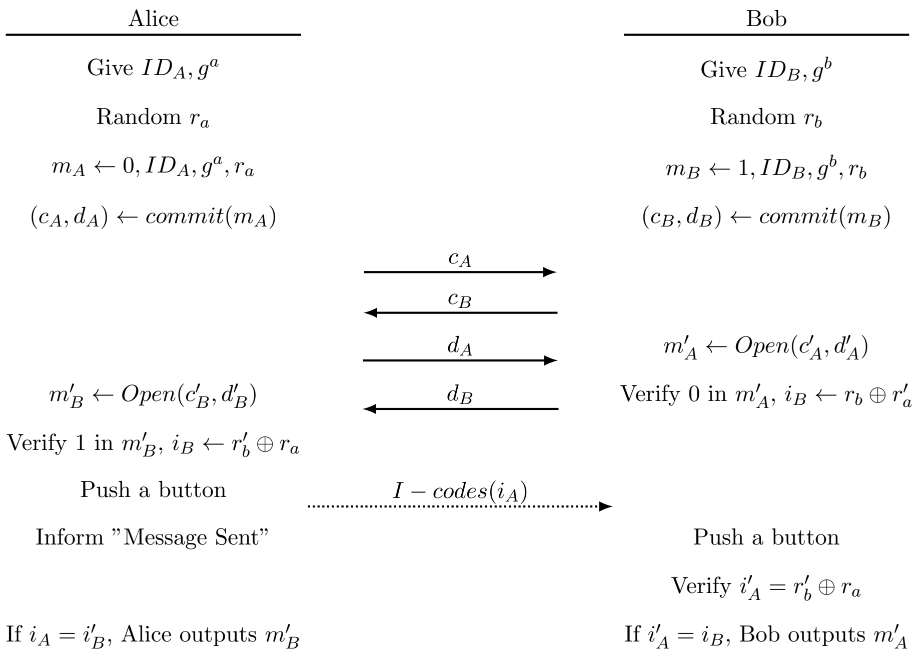
Integrity codes (I-codes) are used with communication channel such that that they are not possible to block signal without being detected. An integrity code is a seven-tuple $(\mathcal{S}, \mathcal{M}, \mathcal{E}, \mathcal{P}, l, t, e_c)$ where:

1. $\mathcal{S}$ is possible of source states.
2. $\mathcal{M}$ is a set of binary of sequence t 1's and $l - t$ 0's.
3. $\mathcal{E}$ is a set of source encoding rules $e_s : \mathcal{S} \to \mathcal{M}$, where $e_s \in \mathcal{E}$ is an injective function.

4. $\mathcal{P}$ is a set consisting of two power levels 0 and p with $p > 0$.

5. $e_c : \mathcal{M} \rightarrow \mathcal{P}^{\updownarrow}$ is a channel modulation function satisfying rules: (i) symbol "1" is transmitted using power level p, symbol "0" is transmitted using power level 0.

Based on a concept of I-codes, its application on DH-SC is straightforward. Bob and Alice complete four steps of DH-SC protocol at beginning. Alice encoded authentication value i_A into $I{-}code$, then transmits it in an out-of-band channel to Bob. After reception of $I - code$, Bob probably ensures that this $I - code$ is sent from Alice, and verifies if authenticated value i'_A is equal to i_B or not.


Alice
Bob
Give ID_A, g^a
Give ID_B, g^b
Random r_a
Random r_b
$m_A \leftarrow 0, ID_A, g^a, r_a$
$m_B \leftarrow 1, ID_B, g^b, r_b$
$(c_A, d_A) \leftarrow commit(m_A)$
$(c_B, d_B) \leftarrow commit(m_B)$
c_A
c_B
d_A
$m'_A \leftarrow Open(c'_A, d'_A)$
$m'_B \leftarrow Open(c'_B, d'_B)$
d_B
Verify 0 in m'_A, $i_B \leftarrow r_b \oplus r'_a$
Verify 1 in m'_B, $i_B \leftarrow r'_b \oplus r_a$
Push a button
$I - codes(i_A)$
Inform "Message Sent"
Push a button
Verify $i'_A = r'_b \oplus r_a$
If $i_A = i'_B$, Alice outputs m'_B
If $i'_A = i_B$, Bob outputs m'_A

1.2.0.10 Short Random String-Based Key Agreement Protocol

Authors in [9] claimed that SAS-based AKA methods strongly required assistance from users. Therefore, they proposed a scheme, namely Short Random String (SRS)-based key agreement protocol, automatically pairing two devices using an audio channel. The proposed scheme was claimed more advantage than SAS-based scheme in term of bandwidth utilisation.

The 3-move protocol between *Alice* and *Bob* is as follow.

1. A device A sends A's public key PK_A to device B.

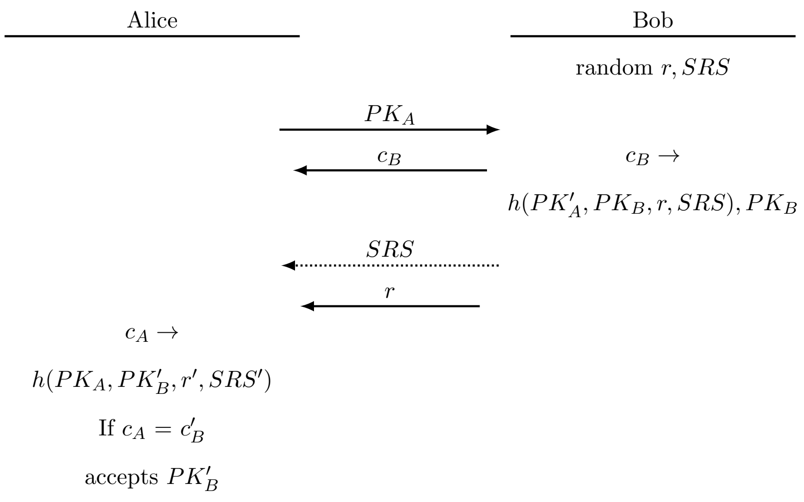
2. A device B generates two random strings, r and SRS. Then B calculates a hashed value using PK_A, PK_B, SRS and r and sends this value to A.

3. B sends SRS to A over an out-of-band channel.

4. B sends r to A.

5. A verifies the hashed value received in step 2 using PK_A, PK_B, SRS and R. If it is successful, A generates agreed key using authenticated B's public key.

In the protocol, the verification process is only decided by A, then attackers easily impersonate A's public key. However, this attack can be prevented by human monitoring or confirming from $A$to B.



Strength of the protocol completely lies on length of SRS string. If the length of SRS message is p, then success probability of attack is less than or equal 2^{-p}.

Summing up this part is presented at the table 1.5.

1.3 2-Move Secure Device Pairing Protocol

In this section, we propose a novel pairing protocol. This proposal is relatively efficient in sense that it only requires 2 messages on a wireless channel plus one message on a unidirectional public OOB channel. In spite of this, we will see in section 1.3.1 that it still preserves an attack success probability of 2^{-k}, where k is length of random numbers. The picture 1.1 describes how the protocol works.

The protocol is depicted in Figure 1.1. It runs between the initiator Alice (A) and the responder Bob (B), who intend to securely exchange their public keys denoted as g^a and g^b respectively. The protocol is presented as below.

1. Alice picks a random value r_a. Bob picks a random value r_b
2. Alice sends $(g^a, h(g^a, r_a))$ to Bob.
3. Bob sends (g^b, r_b) to Alice.
4. Alice sends $(r_a \oplus h_{r_b}(g^a, g^b))$ to Bob over a public channel.
5. Bob verifies received value and announces the result to Alice.
6. Alice confirms by pushing an Accept button.

Random values r_a, r_b and use of hash function in the protocol work as an instance of a commitment scheme whose design has strong results on minimising attack probability as we will see in section 1.3.1.

This protocol achieves strong security even with a low-bandwidth OOB channel. Alice and Bob are assumed to be honest or uncompromised. After receiving the third message Bob can perform verification and then securely notify Alice of an outcome with the human user's help.

Table 1.6 summarises a comparison of our protocol with main existing protocols exploiting a public out-of-band channel. We took into account a number of messages on wireless channel, a number of messages on OOB channel, computation cost, and existence of a formal proof of security (either in the Dolev-Yao model or computational model). In the table, H (resp. MAC, XOR, C) denotes an application of a hash function (resp. a message authentication code, an exclusive-or, a commitment scheme).

In term of computation complexity, our protocol can catch up with other competitors since one exclusive-or operation on a short string does not significantly impact the total cost. Concerning communication cost, our protocol uses the least number of messages. Furthermore, a security analysis has been performed both in Dolev-Yao and computational models.

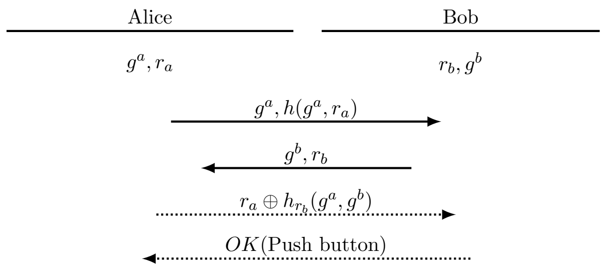


FIGURE 1.1: New 2-move Authenticated Key Agreement Protocol

1.3.1 Analysis In Computational Model

We give a sketch of proof of our protocol in a computational model. Our goal is to evaluate the successful attacking probability against the protocol. Similarly to [42] we conduct an analysis based on the model presented in [43].

We refer to the security definition in [42] which is presented as follows.

Definition 1.3.1. We say that a protocol is a secure protocol enabling authentication of DH public parameter between A and B if attacker cannot succeed in deceiving A and B into accepting DH public parameters different then g^{xa} and g^{xb}, except with a satisfactorily small probability $\mathcal{O}(2^{-k})$.

Lemma 1.3.2. For any interaction between strand st and st', and attacker X, attack success probability of X is lower or equal to $n.\gamma.2^{-k}$, where n is the number of participants on the network, γ is the maximum number of sessions for each participant, k is the length of short authenticated string.

Proof. For a normal run, the responder uses value of r_a extracted from messages received on OOB channel to open the commitment $h(g^a, r_a)$. If the responder opens successfully, an Accept statue is notified. So, to win the game, attacker X has to deliver $h(m)$ in the first message such that $h(m) = h(g^{xa}, (r_a \oplus h_{r_{xb}}(g^a, g^{xb})) \oplus (h_{r_b}(g^{xa}, g^b)))$ (1) for some m. Due to our assumption on MAC function equation (1) can be rewritten in $h(m) = h(g^{xa}, (r_a \oplus r_{xb} \oplus r_b \oplus h(g^a, g^{xb}) \oplus h(g^{xa}, g^b)))$ (2).

In the simplest case where attacker X is able to find a pair (g^{xa}, g^{xb}) such that $h(g^{xa}, g^b) = h(g^a, g^{xb})$, we can deduce from (2) to $r_{xa} = r_a \oplus r_{xb} \oplus r_b$ or $r_{xa} \oplus r_b = r_a \oplus r_{xb}$ (3).

Observe that X has to submit r_{xb} before actually knowing r_a. Similarly, X has to submit r_{xa} before actually seeing r_b. Thus irrespectively, the attacking strategy taken by X, r_a and r_b will be revealed after r_{xa} and r_{xb} have been generated and submitted. If it happens that both r_a and r_b are revealed in the same time, then we can pick an arbitrary one.

Assume that r_a is revealed after r_b, we have:

(i) r_a and r_b are independently and uniformly distributed random variables,

(ii) r_{xa} and r_{xb} must be generated and submitted before either r_a is revealed,

(iii) each principal can open at most γ sessions.

The same holds for a case where r_b is revealed after r_a. Therefore, $Pr[r_{xa} \oplus r_b = r_a \oplus r_{xb}] \leq n.\gamma.2^{-k}$.

In a normal case where X cannot find collision, let $t = h(g^a, g^{xb}) \oplus h(g^{xa}, g^b)$ since g^a and g^b are not changed over sessions. We have (i)$r_{xa} \oplus r_b = r_a \oplus r_{xb} \oplus t$, (ii) t could be constant, hence, $Pr[r_{xa} \oplus r_b = r_a \oplus r_{xb} \oplus t] \leq n.\gamma.2^{-k}$. □

1.3.2 An Implemetation on An Embedded System

We produce a prototype of our device pairing protocol using two Arduino boards. Because of lacking WIFI shields, two Ethernet shields are used instead to provide a network connect between two boards. The testing system includes:

- An Arduino Uno equips with an Ethernet shield, and a LED light.
- An Arduino Mega equips with an Ethernet shield, and a light sensor.
- Two boards connect via an Ethernet cable.

The testbed is presented at the figure 1.2. A visual light communication served as an out-of-band communication channel is intuitively established via a LED and a light sensor. Data transmitting in this channel are encoded by a sequence of blinking light. In particular, while the LED hitting the highest brightness represents for a bit ”1”, the LED dims to represent for a bit ”0”. Furthermore, distance between the LED and the light sensor is short enough so that the sensor can correctly recognise bits from the LED. The accuracy of OOB transmission strongly depends on this distance and noise of environment apparently.

All tests have been done in normal lab environment. We ran 10 times, and in each time 32-bits OOB message was transmitted between two boards. Under 3 centimetres, we get 100% successful rate. The rate reduces to 75% in 5 centimetres, and dramatically drops at further distance. However, in low noise environment, the results are remarkably improved. The result of executing time of some functions is presented in the table 1.7. With these results, we believe that our solution is practical.

1.4 Flaws Found in Some Pairing Protocols

In this section, we are going present flaws that we have found in some pairing protocols. These flaws have not revealed in any publication before.

We will explain clearly how come we can discover these attack in the next chapter. Briefly speaking, we constructed a formal model to analyse our protocol, and adapted it

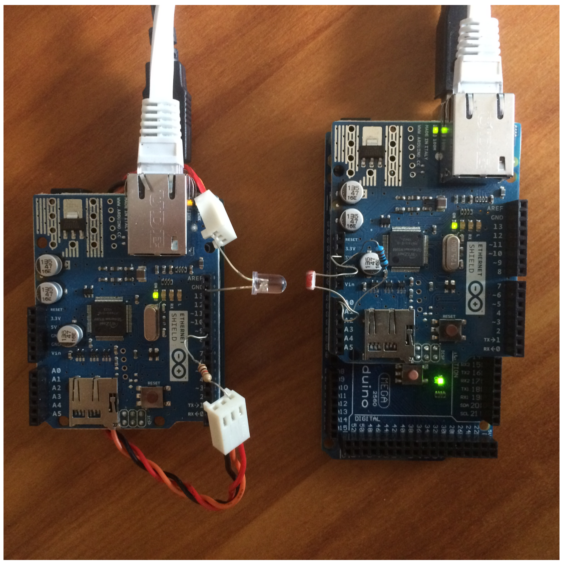


FIGURE 1.2: Prototypes

to do other existing device pairing ones introduced in this chapter. As a result of that, some flaws have been discovered.

To begin with, we would like to take some assumptions on channels, attacker capabilities, and user actions as below.

- Two participants don't play the protocol concurrently.
- The protocol replays when it accidentally gets an error.
- Attacker knows OOB message content before the message is delivered.
- Attacker can delay user's actions.

Furthermore, protocols are considered in theoretical perspective where OOB channels are classified in our category. In particular, these following protocols are assumed to use long-range public out-of-band channels which only provide channel origin authentication.

1.4.1 Attack on Wong-Stajano Protocol using Bidirectional Channel

The Wong-Stajano protocol using bidirectional channel was presented at 1.2.0.7 in this chapter. The protocol aims to provide a key agreement between two participants. However, we found a counterexample in which the protocol goal does not hold for the initiator. The counterexample is illustrated in the figure 1.3, and is precisely detailed in table 1.8.

Additionally, we found the protocol is being used in the Pico system of the authors [44].

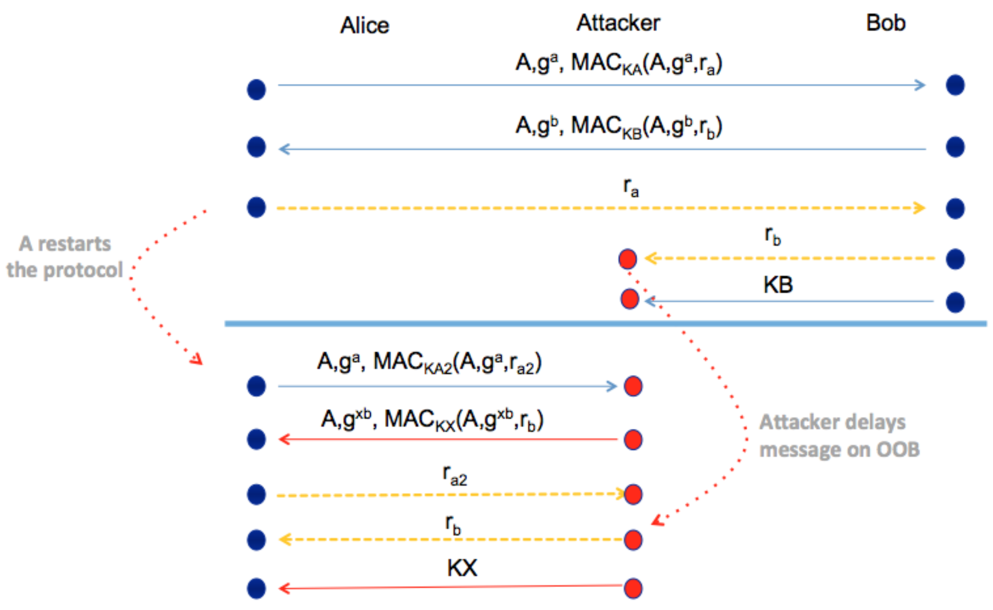


FIGURE 1.3: Attack on Wong-Stajano Protocol using Bidirectional Channel with Unidirectional Channel

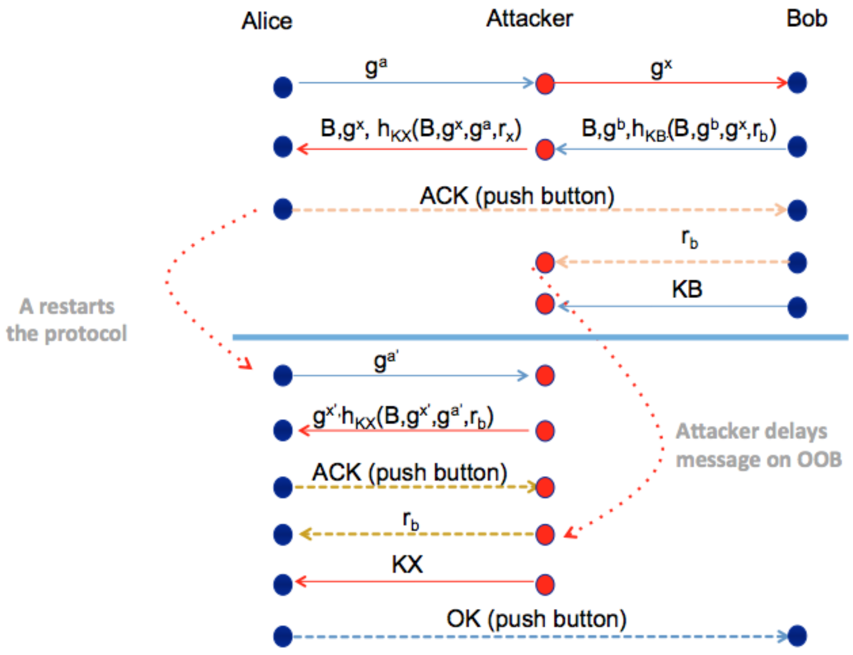


FIGURE 1.4: Attack on Wong-Stajano Protocol using Unidirectional Channel

1.4.2 Attack on Wong-Stajano Protocol using Unidirectional Channel

The counterexample of Wong-Stajano protocol using unidirectional channel is illustrated in the figure 1.9, and is precisely detailed in table 1.9.

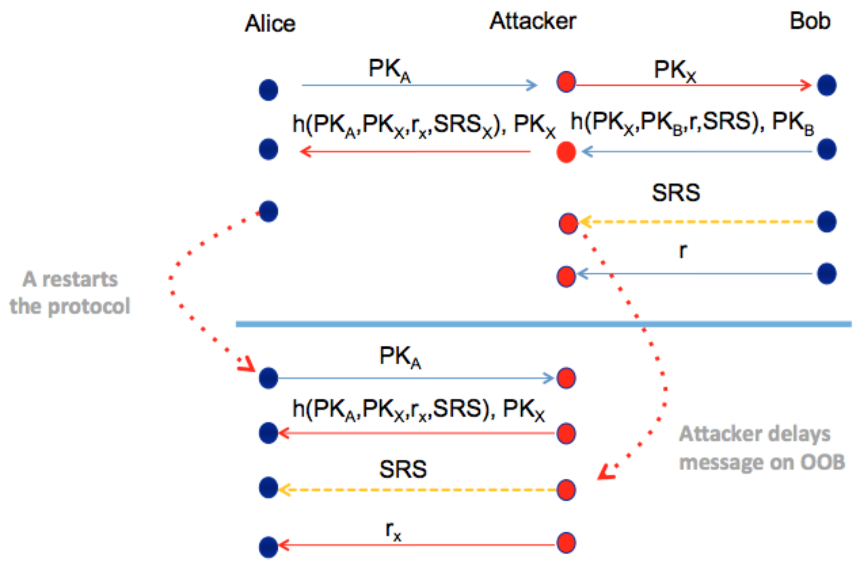


FIGURE 1.5: Attack Against SRS-AKA Protocol

1.4.3 Attack on SRS-AKA Protocol

Attack against SRS-AKA is closely identified with one in Wong-Stajano protocol using unidirectional channels. The attack scenario is illustrated in figure 1.5 and detailed at table 1.10.

1.4.4 Attack on Hoepman AKA Protocol

Since the Hoepman protocol is quite similar to Wong-Stajano protocol with bidirectional channel, the attack found on Wong-Stajano protocol might be used against Hoepman one. But, the difference between two protocols is while two random numbers are exchanged over OOB channel in Wong-Stajano, two short short-hashing values are used in Hoepman version. This apparently lets the attacker a big challenge to break the Hoepman protocol. Nevertheless, due to the less strong hash function, if the attack can find a collision of the short hash function in polynomial time, it is definitely able to launch MITM attack. We take this assumption and present the attack scenario in figure 1.6, and table 1.11.

1.5 Conclusion

In this chapter, we have conducted a deep survey on out-of-band channel types and secure device pairing protocols. Moreover, we have provided a novel solution to the fundamental issues of key agreement over a radio links. To our knowledge, our proposal requiring

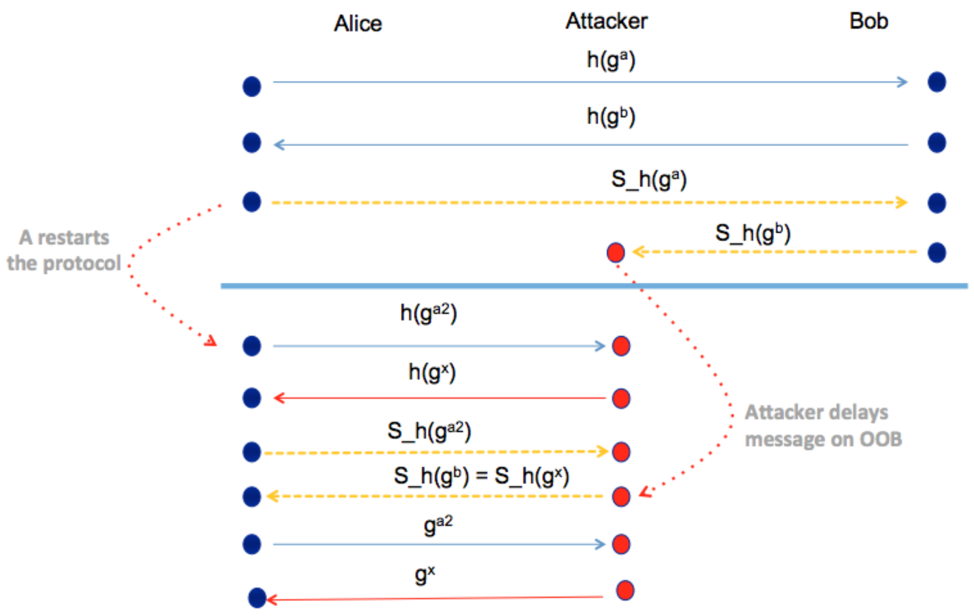


FIGURE 1.6: Attack Against Hoepman-AKA Protocol

only 2 wireless radio messages and one out-of-band message is more economic than existing protocols in term of communication cost. Yet it still provides a security level equivalent to one of existing protocols. We also gave a sketch of proof in a computational model. Additionally, an implement of our protocol has been conducted and tested in an embedded system to show that it is practical via our benchmark. In mean while, we found some flaws of Wong-Stajano protocols, Hoepman protocol, and SRS-AKA protocol, this has not introduced before.

Table 1.5: Paring Method Comparision

Approach	**Number of Message**		**Computation**	**Communication**	**Required Crypto-graphic Primitives**	**Comment**
	Wireless Channel	**OOB Channel**	**Cost per Sid**	**Cost over OOB**		
Maher Manual Authentication	2	2	1 CF	2* 80 bits	CF	Long OOB message
Talking to Strangers	2	2	1 HASH	2 ID + 2 HASH	HF	Not specific HASH output length, long OOB message
VIC	3	1	1 HASH	20 bits HASH	HF	Long OOB message
MANA I	0	1	1 CF	20 bits K + 20bits CV	CF	
MANA II	1	1	1 CF	20 bits K + 20bits CV	CF	
MANA III	4	1	2 * HASH	16-20bits K	MAC	
Wong-Stajano MANA III	6	2	1 MAC	2 *(20 bits N)	HF	
Wong-Stajano Bidirectional Out-of-band Channel	4	2	1 MAC	2 *(20 bits N)	HF	
Wong-Stajano Unidirectional Out-of-band Channel	3	1	1 MAC	20 bits N	HF	
Improved Wong-Stajano	4	1	1 HASH + 1 XOR	1* (20 bits N)	HF + XOR	
Hoepman AKA	4	2	1 L_HASH + 1 S_HASH	2* (n bits S_HASH)	Short(S_) and Long(L_) HF	n is unclear, fresh random DH public keys
Improved Ephemeral Pairing	3	1	1 L_HASH + 1 S_HASH	80 bits S_HASH	Short(S_) and Long(L_) HF	fresh random DH public keys
4-move SAS Vaudenay	4	2	1 CS + 1 XOR	15bits SAS	CS + XOR	
3-move SAS Vaudenay	3	1	1 CS + 1 XOR	15bits SAS	CS + XOR	
SAS Cros AKA Pasini-Vaudenay	3	2	1 CS + 1 XOR + 1 HASH	2 * (15-20bits SAS)	CS + XOR + HF	
MANA IV	3	1	1 CS + 1 HASH	14bits SAS	CS + HF	
MA-DH	3	1	1 CS + 1 HASH	14bits SAS	CS + HF	
SRS-based AK	3	1	1 CS	15 bits SRS	CS	
DH-SC	4	2	2 CS + 1 XOR	15bits SAS	CS +XOR	
DH-DB	6	2	1 CS + 1 XOR	15bits SAS	CS +XOR	

ID	Identification
CS	commitment Scheme
XOR	XOR operation
CV	Check Value
CF	Check Function
HF	Hash Function
N	Nonce
S_HASH	Short Hash
L_HASH	Long Hash

Table 1.6: Device Pairing Protocol Comparison

Protocol	Wireless Message	OOB Message	Computation Cost	Formal Proof
Bidirectional Wong-Stajano [38]	4	2	2*MAC	FAIL
Unidirectional Wong-Stajano [38]	3	2	1*MAC	FAIL
Improved Wong-Stajano [37]	4	1	2*H + 1*XOR	$\emptyset$
DH-SC [42]	4	1	2*C + 1*XOR	$\surd$
4-Move SAS [39]	4	1	2*C + 1*XOR	$\surd$
3-Move SAS [39]	3	1	1*C +1H + 1*XOR	$\surd$
MANA IV [41]	3	1	1*C +1*H	$\surd$
MA-DH [41]	3	1	1*C +1*H	$\surd$
SRS [9]	3	1	1*H	FAIL
Our Proposal	2	1	1*H +1*MAC + 1*XOR	$\surd$

Table 1.7: Performance Evaluation of Device Pairing Protocol

Function	**Time(ms)**
Generating Keys	3898
commitment Calculation	15
commitment Validation	15
Transferring OOB message	3201
Complete Protocol	21760

Table 1.8: Attack Scenario Against Initiator's Guarantee in Wong-Stajano Protocol with Bidirectional Channel

Step 1.1	Attacker suspends the r_b sent by Bob on OOB channel
Step 1.2	Attacker drops the KB sent by Bob.
Step 1.3	Attacker starts a new session with Alice
Step 2.1	Alice sends $(A, g^a, MAC_{KA2}(A, g^a, R_{A2}))$ to the Attacker on wireless channel
Step 2.2	Attacker sends $(B, g^x, MAC_{KX}(B, g^x, R_B))$ on wireless channel
Step 2.3	Attacker drops R_{A2} sent by Alice on OOB channel
Step 2.4	Attacker releases r_b at Step 1.1 on OOB channel
Step 2.5	Attacker sends KX to Alice on wireless channel
Step 2.6	At the end of the execution, Alice believes she shares a fresh session key with Bob, known actually by the Attacker

Table 1.9: Attack Scenario Against Initiator's Guarantee in Wong-Stajano Protocol with Unidirectional Channel

Step 1.1	Attacker intercepts g^a sent by Alice on wireless channel
Step 1.2	Attacker replies with $(B, g^x, h_{k_X}(B, g^x, g^a, r_x))$ to Alice on wireless channel
Step 1.3	Attacker suspends r_b sent by Bob on OOB channel, and starts a new session with Alice
Step 2.1	Alice sends $g^{a'}$ on wireless channel
Step 2.2	Attacker responds $(B, g^x, h_{k_X}(B, g^{a'}, g^{x'}, r_b))$ on wireless channel
Step 2.3	Attacker drops ACK sent by Alice on OOB channel
Step 2.4	Attacker release r_b sent by Bob on OOB channel at Step 1.3
Step 2.5	Attacker sends k_X to Alice on Wireless channel
Step 2.6	At the end of the execution, Alice believes she shares a fresh session key with Bob, known actually by the Attacker

Table 1.10: Attack Scenario Against Initiator's Guarantee in SRS-AKA Protocol

Step 1.1	Attacker intercepts PK_A sent by Alice on wireless channel
Step 1.2	Attacker replies with $h(PK_A, PK_X, r_x, SRS_X), PK_X$ to Alice on wireless channel
Step 1.3	Attacker suspends SRS sent by Bob on OOB channel, and starts a new session with Alice
Step 2.1	Alice sends PK_A on wireless channel
Step 2.2	Attacker responds $h(PK_A, PK_X, r_x, SRS), PK_X$ on wireless channel
Step 2.4	Attacker release SRS sent by Bob on OOB channel at Step 1.3
Step 2.5	Attacker sends r_x to Alice on Wireless channel
Step 2.6	At the end of the execution, Alice believes she shares a fresh session key with Bob, known actually by the Attacker

Table 1.11: Attack Scenario Against Initiator's Guarantee in Hoepman Protocol

Step 1.1	Attacker suspends the $S\_h(g^b)$ sent by Bob on OOB channel
Step 1.2	Attacker finds g^x so that $S\_h(g^x) = S\_h(g^b)$. It does not necessary to find $h(g^x) = h(g^b)$
Step 1.3	Attacker starts a new session with Alice
Step 2.1	Alice sends $h(g^{a2})$ to the Attacker on wireless channel
Step 2.2	Attacker sends $h(g^x)$ to Alice on wireless channel
Step 2.3	Attacker drops $S\_h(g_{a2})$ sent by Alice on OOB channel
Step 2.4	Attacker releases $S\_h(g^b)$ at Step 1.1 on OOB channel
Step 2.5	Alice sends g^{a2} to Attacker on wireless channel
Step 2.6	Attacker sends g^x to Alice on wireless channel
Step 2.7	At the end of the execution, Alice believes she shares a fresh session key with Bob, known actually by the Attacker

Appendix A

Strand Spaces Model

In 1997, Fabrega, Herzog and Gutman developed a new method to prove a protocol if it achieves authentication and secrecy properties or not. First published internally as a technical report, the work went out public in a conference paper [45] in a year later. Following this approach, the authors continued maturing their model. Basic Strand Spaces theory was extended with *honest idea* [46] which allows to learn general principles that limit penetrators' capabilities. After that, *mixed strands* [47] was proposed to study problems of mixed protocols. One huge milestone of Strand Spaces theory is *authentication tests* [48] proposed in 2000. In a study of cryptographic protocols, authors realized that after emitting a message containing a new value, a participant receives a cryptographic form of this value, this must exist a regular participant of the protocol sent it. This scheme, so called an authentication test, works as a powerful tool to minimise work of proving, and straightforwardly give results of authentication protocols. From 2002 to 2007, Strand Spaces theory was enriched with *shape of bundle, skeleton and homomorphism* [49]. These theories study about a sort of protocols that share the same forms. Current developments of Strand Spaces are focusing on various of types of protocols such as TLS [50], location-awareness protocols [2], and DH protocols [51].

Definitions used in thesis are referred from [45], [48] and [49]. Furthermore, to make readers conformable with the theory, the Needham-Schroeder(NS) protocol [52] presented below is used as an example through this part.

1. $A \rightarrow B$: $\{N_a, A\}_{K_B}$
2. $B \rightarrow A$: $\{N_a, N_b, B\}_{K_A}$
3. $A \rightarrow B$: $\{N_b\}_{K_B}$

A.1 Fundamental Theory

A security protocol is an ordered sequence of messages that participants exchange in a protocol. Let $\mathcal{A}$ a set of possible messages intentionally transferred in a security protocol, and elements of this set are referred to as *terms*. Additionally, the set $\mathcal{A}$ is algebra freely generated from of a set of text terms $\mathcal{T}$ and a set of cryptographic keys $\mathcal{K}$ by means of concatenation and encryption. While $\mathcal{T}$ contains textual information such as nonces, $\mathcal{K}$ contains symmetric and/or asymmetric cryptographic keys. The two sets $\mathcal{T}$ and $\mathcal{K}$ are disjoint.

Definition A.1.1. *Compound terms* are generated by two operators

- encr: $\mathcal{K} \times \mathcal{A} \to \mathcal{A}$ representing encryption
- join: $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$ representing concatenation

For convention, from now we express the concatenation of two term t and t' as *t,t'* and encryption of term t with key K as $\{t\}_K$. We continue defining a relation, *subterm relation*, on the set $\mathcal{A}$.

Definition A.1.2. The *subterm relation* $\sqsubseteq$ over $\mathcal{A}$ is defined inductively as:

- $a \sqsubseteq t$ for $t \in \mathcal{T}$ iff $a = t$
- $a \sqsubseteq key$ for $t \in \mathcal{K}$ iff $a = key$
- $a \sqsubseteq gh$ iff $a \sqsubseteq gh, a \sqsubseteq h$ or $a = h$
- $a \sqsubseteq \{g\}_K$ iff $a \sqsubseteq g$ or $a = \{g\}_K$

Remark that, $t_1 \sqsubseteq t$ means t_1 is a subterm of t. A *subterm* is just a term that can be easily extracted from a term with an appropriate key. In NS protocol, initiator A starts sending to intentional responder B a message (or a *term*) of the form $\{N_a, A\}_{K_B}$ where K_B is the public key of B. Subterms of term $\{N_a, A\}_{K_B}$ could be N_a, A, or $\{N_a, A\}_{K_B}$. Since K_B is a key, $K_B \not\sqsubseteq \{N_a, A\}_{K_B}$, except a case when K_B is in a value of this message.

Terms are extended to *signed term* including a positive represented to a transmission, and a negative one represented to a reception.

Definition A.1.3. A *signed term* is a pair $\langle \delta, a \rangle$ with $a \in A$ and δ one of the symbols $+, -$. We will write a signed term as $+t$ or $-t$. $(\pm A)^*$ is the set of finite sequences of signed terms. We denote a typical element of $(\pm A)^*$ by $\langle \delta_1, a_1 \rangle, ..., \langle \delta_n, a_n \rangle$.

Note that, the unsigned term is the term without direction. For example, a signed term $+t$ has direction $+$, and an unsigned term t.

A sequence of sending and receiving messages in a protocol is called *strand*, and a set of strands is called *Strand Spaces*.

When modeling a protocol, the strand of a principal is a sequence of events as seen by that principal in a particular protocol run and in a particular instance of role. For example, the strand of the initiator for the NS protocol is $\langle +\{N_a, A\}_{K_B}, -\{N_a, N_b\}_{K_A}, +\{N_b\}_{K_B}\rangle$. The first event is emission of $\{N_a, A\}_{K_B}$ followed by reception of $\{N_a, N_b\}_{K_A}$ and so on.

Definition A.1.4. A *Strand Spaces* is a set Σ with a trace mapping tr: $\Sigma \to (\pm A)^*$.

For a Strand Spaces Σ:

- A node is a pair $\langle s, i\rangle$, with $s \in \Sigma$ and i an integer satisfying $1 \leq i \leq \mathit{length}(tr(s))$. The set of nodes is denoted by $\mathcal{N}$. We will say the node $n = \langle s, i\rangle$ belongs to the strand s. Clearly, every node belongs to a unique strand.
- If $n = \langle s, i\rangle \in \mathcal{N}$ then $\mathit{index}(n) = i$ and $\mathit{strand}(n) = s$. Define $\mathit{term}(n)$ to be $(tr(s))_i$, i.e. the i^{th} signed term in the trace of s. Similarly, $\mathit{uns\_term}(n)$ is $((tr(s))_i)_2$, i.e. the unsigned part of the i^{th} signed term in the trace of s
- If $n, n' \in \mathcal{N}, n \to n'$ mean $\mathit{term}(n) = +a$ and $\mathit{term}(n') = +a$ It means that node n sends the message a which is received by n', creating a causal link between their strands.
- If $n, n' \in \mathcal{N}, n \Rightarrow n'$ mean n and n' occur on the same strand with $\mathit{index}(n) = \mathit{index}(n') - 1$. It expresses that n is an immediate causal predecessor of n' on the strand.
- $n \Rightarrow^+ n'$ to mean that n precedes n' (not necessarily immediately) on the same strand.
- $m \Rightarrow^* n$ means $m \to n_1 \Rightarrow m_2 \to n_2 \Rightarrow ......m_k \to n$ where n_i, m_i, and n stand on the same or different strands. A *path* created by $m \Rightarrow^* n$ forms a connectivity subgraph in the bundle.
- An unsigned term t *occurs* in $n \in \mathcal{N}$ if $t \sqsubseteq \mathit{term}(n)$.
- An unsigned term t *originates* on $n \in \mathcal{N}$ iff:$\mathit{term}(n)$ is positive, $t \sqsubseteq \mathit{term}(n)$, and whenever n precedes n on the same strand, $t \not\sqsubseteq \mathit{term}(n)$.
- An unsigned term t is *uniquely originating* iff t originates on a unique $n \in \mathcal{N}$

Let the Initiator strand be $st = \langle +\{N_a, A\}_{K_B}, -\{N_a, N_b\}_{K_A}, +\{N_b\}_{K_B}\rangle$, and the responder strand be $st' = \langle -\{N_a, A\}_{K_B}, +\{N_a, N_b\}_{K_A}, -\{N_b\}_{K_B}\rangle$.

The first node of Initiator strand is $\langle st, 1\rangle = +\{N_a, A\}_{K_B}$. Moreover, $\langle st, 1\rangle \Rightarrow \langle st, 2\rangle$ means node $\langle st, 1\rangle$ is a causal predecessor of node $\langle st, 2\rangle$. N_a occurs in $term(\langle st, 1\rangle)$. Additionally, since sign of $\langle st, 1\rangle$ is positive, and there does not exist any predecessor of $\langle st, 1\rangle$, N_a uniquely originates at $\langle st, 1\rangle$.

Let $\mathcal{N}$ be a set of nodes, and let $\mathcal{E}$ be the union of the sets of $\rightarrow$ and $\Rightarrow$ edges. A directed graph $\mathcal{G}$ has a structure $\mathcal{G} = \langle \mathcal{N}, \mathcal{E}\rangle$. A *bundle* is a finite subgraph of this graph in which the edges express casual dependencies of the nodes.

Definition A.1.5. Suppose $\mathcal{N}_{\mathcal{B}}$ be a subset of $\mathcal{N}$, and $\mathcal{E}_{\mathcal{B}}$ be a subset of $\mathcal{E}$. Let $\mathcal{B} = \langle \mathcal{N}_{\mathcal{B}}, \mathcal{E}_{\mathcal{B}}\rangle$ be a subgraph of $\mathcal{G}$. $\mathcal{B}$ is a *bundle* if :

1. $\mathcal{N}_{\mathcal{B}}$ and $\mathcal{E}_{\mathcal{B}}$ are finite
2. If $n \in \mathcal{N}_{\mathcal{B}}$ and $term(n)$ is negative, then there is a unique n' such that $n' \rightarrow n \in \mathcal{E}_{\mathcal{B}}$
3. If $n \in \mathcal{N}_{\mathcal{B}}$ and $n \Rightarrow n'$ then $n \Rightarrow n' \in \mathcal{E}_{\mathcal{B}}$
4. $\mathcal{B}$ is acyclic

The graph consisting of the Initiator strand st and responder strand st' is called a NS bundle. Remarking that Guttman implicitly expressed that when a node transmits a message, there is more than one (or none) receiving node of the message. However, we hardly see more than two receptions in their studies.

Definition A.1.6. A node n belongs to a bundle $\mathcal{B} = \langle \mathcal{N}_{\mathcal{B}}, \mathcal{E}_{\mathcal{B}}\rangle$, written $n \in \mathcal{B}$ if $n \in \mathcal{N}_{\mathcal{B}}$. The $\mathcal{B} - height$ of a strand $s \in \mathcal{B}$ is the largest i such that $\langle s, i\rangle \in \mathcal{B}$.

For instance, in NS protocol, $\mathcal{B} - height$ of initiator strand is 3, similarly $\mathcal{B} - height$ of responder strand is also 3.

A.2 Component, Authentication Tests

After introducing Strand Spaces fundamental theory, basing on the fact that security authentication protocols usually use specific challenge-response methods to obtain goals, Thayer et al[53] continued publishing a theory of *authentication tests* as a tool to prove security properties for protocols easily.

Definition A.2.1. A term t' is called a *component* of term t if t' cannot be split to another term t'', and t is built by concatenating t' with arbitrary terms.

Conveniently, we refer the example of Thayer[2]. If we have a term like $B\{N_aK\{KN_b\}_{K_B}\}_{K_A}N_a$, then it contains three components: $B, \{N_aK\{KN_b\}_{K_B}\}_{K_A}$, and N_a.

Definition A.2.2. For a strand s, a term t is *new* at $n = \langle s, i\rangle$ if t is a component of term(n), but t is not a component of node $\langle s, j\rangle$ for every $j < i$.

Taking NS protocol as an example, the component $\{N_a, N_b, B\}_{K_B}$ is new at node $\langle st, 2\rangle$ since it is clearly not a component of $\langle st, 1\rangle$.

Definition A.2.3. Suppose that $n \in \mathcal{B}$ is positive, $a \in \mathcal{A}$ is a subterm of $term(n)$. The edge $n \Rightarrow^+ n'$ is a *transformed edge* for a if there exits a negative node $n' \in \mathcal{B}$, and there is a new component t_2 of n' such that $a \sqsubseteq t_2$.

Respectively, *a transforming edge* is denoted.

Definition A.2.4. Suppose that $n \in \mathcal{B}$ is negative, $a \in \mathcal{A}$ is a subterm of $term(n)$. The edge $n \Rightarrow^+ n'$ is a *transforming edge* for a if there exits a positive node $n' \in \mathcal{B}$, and there is a new component t_2 of n' such that $a \sqsubseteq t_2$.

For example, the edge $\langle st, 1\rangle \Rightarrow \langle st, 2\rangle$ is a transformed edge for the term N_a, and $\langle st', 1\rangle \Rightarrow \langle st', 2\rangle$ is a transforming edge for N_a.

Definition A.2.5. The edge $n \Rightarrow^+ n'$ is *a test* for $a \in \mathcal{A}$ if a uniquely originates at n, and $n \Rightarrow^+ n'$ is a transformed edge for a.

The transformed edge $\langle st, 1\rangle \Rightarrow \langle st, 2\rangle$ is a test for N_a since N_a uniquely originates at $\langle st, 1\rangle$.

Definition A.2.6. Suppose that $n, n' \in \mathcal{B}$.

1. The edge $n \Rightarrow^+ n'$ is *a outgoing test* for $a \sqsubseteq t = \{h\}_K$ if it is a test for a in which $K^{-1} \notin P$, a does not occur in any component of n other than t. Moreover, t is a test component for a in n.

2. The edge $n \Rightarrow^+ n'$ is *a incoming test* for $a \sqsubseteq t_1 = \{h\}_K$ if it is a test for a in which $K \notin P$, and t_1 is a test component for a in n'.

The transformed edge $\langle st, 1\rangle \Rightarrow \langle st, 2\rangle$ could be considered as an incoming test for N_a.

Subsequently, authentication tests [53] are provided as powerful and simple tools to guarantee existence of regular strands in a bundle.

Authentication Test 1: Suppose that $n' \in \mathcal{B}$, and $n \Rightarrow^+ n'$ is outgoing test for $a \sqsubseteq t$ with $t = term(n)$. Then there exist regular nodes $m, m' \in \mathcal{B}$ such that t is a component of m, and $m \Rightarrow^+ m'$ is a transforming edge for a. In addition that a occurs only in component

$t_1 = \{h_1\}_{K_1}$ of m', that t_1 is not a proper subterm of any regular component, and that $K_1^{-1} \notin P$. There is a negative regular node with t_1 as a component.

Authentication Test 2: Suppose that $n \in \mathcal{B}$, and $n \Rightarrow^+ n'$ is incoming test for $a \sqsubseteq t'$ with $t' = term(n')$. Then there exist regular nodes $m, m' \in \mathcal{B}$ such that t' is a component of m', and $m \Rightarrow^+ m'$ is a transforming edge for a.

Definition A.2.7. A negative node is an *unsolicited test* for $t = \{h\}_K$ if t is a test component for any a in n and $K \notin P$.

Authentication Test 3: Suppose that a node n is in a bundle $\mathcal{B}$, and n be an unsolicited test for $t = \{h\}_K$, then there exists a positive regular node $m \in \mathcal{B}$ such that t is a component of m.

The proofs of these authentication tests are out of scope in this part. So if readers eager to deeply understand the proofs, please regard to the paper [48].

A.3 Shape and Skeleton

In design protocol, ones always desire that there is only one possible execution of their protocol in any scenario. Nevertheless, there might exist some executions relative to the protocol's assumptions. Actually, the executions of protocols normally have very few essentially different forms that called *shapes*. Then, authentication and secrecy properties can be determined by examining the shapes.

Precisely, a shape is a local execution by honest principals. Partial information about a principal's execution of a protocol is called *skeleton*. Skeletons are partial-ordered structures, or fragments of message sequence chart. Moreover, a skeleton is *realised* if it is not fragmented, i.e. it contains exactly the regular behavior of some executions. A realised skeleton is a shape if it is minimal.

A preskeleton describes the regular parts of a set of bundles. Formally presented, preskeleton is defined as follows.

Definition A.3.1 (Skeleton). A four-tuple $\mathcal{A} = (node, \preceq, non, unique)$ is a preskeleton if:

1. $node$ is a finite set of regular nodes, $n_1 \in node$ and $n_0 \Rightarrow^+ n_1$ implies $n_0 \in nodes$;
2. $\preceq$ is a partial ordering on $node$ such that $n_0 \Rightarrow^+ n_1$ imples $n_0 \preceq n_1$;
3. non is a set of keys where if $K \in non$, then for all $n \in node, K \not\sqsubseteq term(n)$, and for some $n' \in node$, either K or K^{-1} is used in $term(n')$;

4. *unique* is a set of atoms where $a \in unique$, for some $n \in node, a \sqsubseteq term(n)$.

A preskeleton $\mathcal{A}$ is a *skeleton* if in addition:

4'. $a \in unique$ implies a originates at no more than one $n \in node$.

Definition A.3.2 (Shape). $\mathcal{A}'$ is a *shape* for $\mathcal{A}$ if (1) some $H : \mathcal{A} \rightarrow \mathcal{A}'$, (2) $\mathcal{A}'$ is realised, and (3) no proper skeleton of (A') satisfies (1) and (2).

A strand could be considered as a skeleton. And a bundle could be considered as a shape. Furthermore, a bundle containing a penetrator strand is also a possible shape of the protocol.

A.4 Penetrator Model

Original Strand Spaces theory uses Dolev-Yao [54] model as its penetrator model. Penetrator's power is built up from two ingredients: initially known keys available to the penetrator, and actions that allows the penetrator manipulates messages. The actions are summarized to discard message, to generate arbitrary messages, to concatenate messages together, and to apply cryptographic operation using available keys. The model is described as follows.

Definition A.4.1. A penetrator trace is *one of the following*:

M. Text message: $\langle +t \rangle$ where $t \in T$

F. Flushing: $\langle -g \rangle$

T. Tee: $\langle -g, +g, +g \rangle$

C. Concatenation $\langle -g, -h, +gh \rangle$

S. Separation into components: $\langle -gh, +g, +h \rangle$

K. Key: $\langle +K \rangle$ where $K \in \mathcal{K}_{\mathcal{P}}$

E. Encryption: $\langle -K, -h, +\{h\}_K \rangle$

D. Decryption: $\langle -K^{-1}, -\{h\}_K, +h \rangle$

This penetrator's trace set given here could be extended if desired without any modification on the whole model. However, the proofs should be adjusted to take into account the additional penetrator traces. This ability gives us an open space to add some physical penetrator traces without worry of proving way.

Appendix B

Analysis Wong-Stajano Protocol in AVISPA

B.1 Transformed Protocol

```
%Wong Stajano Protocol - OOB Transformation
%( F: hash function, Ks: pre-shared key,
%ACK: confirm message)
%( attacker knowledge: G, Hash, A, B)

A->B : Ga
B->A : Gb.h(Ga.Gb.Rb.Kb)

A->B : Rat.{F(ACK.Rat)}_Ks
B->A : Rbt.F(Rat.Rbt)
A->B : ACK.{F(ACK.Rat.Rbt)}_Ks

B->A : Rb.{4.F(Rb.Ks)}_Ks

B->A : Kb
A->B : {ACK.1.A.B}_Ks %confirm acceptance

%verify keys
SK = exp(Gb,Ea) or SK = exp(Ga,Ea)
A->B : {A.Rat3}_SK
B->A : {B.Rat.Rb3}_SK
A->B : {B.Rbt3}_SK
```

B.2 Source Code

```
%%Wong Stajano Protocol - OOB Transformation
% ( F: hash function, Ks: pre-shared key, ACK: confirm message)
% ( attacker knowledge: G, Hash, A, B)
%
% A->B : Ga
% B->A : Gb.h(Ga.Gb.Rb.Kb)
% A->B : Rat.{3.F(ACK.Rat)}_Ks
% B->A : Rbt.F(3.Rat.Rbt)
% A->B : ACK.{3.F(ACK.Rat.Rbt)}_Ks
% B->A : Rb.{4.F(Rb.Ks)}_Ks
% B->A : Kb
% A->B : {ACK.1.A.B}_Ks %confirm acceptance
%
% verify keys
% SK = exp(Gb,Ea) or SK = exp(Ga,Ea)
% A->B : {A.Rat}_SK
% B->A : {B.Rat.Rbt}_SK
% A->B : {B.Rbt}_SK
%

role alice(A, B : agent,
          G: text,
      ACK: message,
      F: hash_func,
          Ks: symmetric_key,
          SND,RCV: channel(dy))

played_by A def=
        local State   : nat,
        Ea: text,
        Ga, Gb: message,
        Kb,SK: symmetric_key,
        Rb, Rat, Rbt : text,
         Commit: hash(message. message.text.symmetric_key),
        Hb: hash(text.symmetric_key)

init State := 0

transition

1. State = 0 /\ RCV(start)
   =|>
   State' := 2 /\ Ea' := new()
               /\ Ga' := exp(G,Ea')
               /\ SND(Ga')

2. State = 2 /\ RCV(Gb'.Commit')
   =|>
   State' := 4 /\ Rat' := new()
         /\ SND(Rat'.{3.F(ACK.Rat')}_Ks)

3. State = 4 /\ RCV(Rbt'.Hb')
             /\ Hb' = F(3.Rat.Rbt')
   =|>
   State' := 6 /\ SND(ACK.{3.F(ACK.Rat.Rbt')}_Ks)



4. State = 6 /\ RCV(Rb'.{4.Hb'}_Ks)
             /\ Hb' = h(Rb'.Ks)
             /\ RCV(Kb')
         /\ Commit = F(Ga.Gb.Rb'.Kb')
   =|>
State' := 8 /\ SND({ACK.1.A.B}_Ks)
              /\ SK' := exp(Gb,Ea)
              /\ Rat':= new()
              /\ SND({A.Rat'}_SK)
              /\ secret(Rat',rat,{A,B})
              /\ witness(A,B,bob_alice_rat,Rat')

5. State = 8 /\ RCV({B.Rat.Rbt'}_SK)
   =|>
   State':= 10 /\ SND({B.Rbt'}_SK)
              /\ request(A,B,alice_bob_rbt,Rbt')

end role

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

role bob( B, A : agent,
          G: text,
          ACK: message,
          F: hash_func,
          Ks: symmetric_key,
          SND,RCV: channel(dy))

played_by B def=
        local State   : nat,
        Eb: text,
        Ga, Gb: message,
        Kb,SK: symmetric_key,
        Rb, Rat, Rbt : text,
         Commit: hash(message. message.text.symmetric_key),
        Ha: hash(message.text),
    Ha2: hash(message.text.symmetric_key)



init
        State := 1

transition

1. State = 1 /\  RCV(Ga')
   =|>
   State' := 3 /\ Rb' := new()
               /\ Kb' := new()
               /\ Eb' := new()
               /\ Gb' := exp(G,Eb')
           /\ SK' := exp(Ga',Eb')
           /\ Commit' := F(Ga'.Gb'.Rb'.Kb')
               /\ SND(Gb'.Commit')

2. State = 3 /\ RCV(Rat'.{3.Ha'}_Ks)
   =|>
   State' := 5 /\ Rbt' := new()
                   /\ SND(Rbt'.F(3.Rat'.Rbt'))

3. State = 5   /\ RCV(ACK.{3.Ha2'}_Ks)
               /\ Ha2' = F(ACK.Rat.Rbt)
               /\ Ha = F(ACK.Rat)
   =|>
   State' := 7 /\ SND(Rb.{4.F(Rb.Ks)}_Ks)
               /\ SND(Kb)

4. State = 7   /\ RCV({ACK.1.A.B}_Ks)
               /\ RCV({A.Rat'}_SK')
   =|>
   State' := 9 /\ Rbt' := new()
              /\ SND({B.Rat'.Rbt'}_SK)
              /\ secret(Rbt',rbt,{A,B})
              /\ witness(B,A,alice_bob_rbt,Rbt')
```

```
5. State = 9  /\ RCV({B.Rbt}_SK)
   =|>
   State' := 11 /\ request(B,A,bob_alice_rat,Rat)

end role

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

role session(A,B : agent,
             G: text,
         ACK: message,
             F: hash_func,
             Kab: symmetric_key)

def=
  local SA, RA, SB, RB: channel (dy)

  composition
     alice(A,B,G,ACK,F,Kab,SA,RA)
  /\ bob  (B,A,G,ACK,F,Kab,SB,RB)

end role
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment() def=


    const a, b          : agent,
        g                   : text,
        ack                    : message,
        h           : hash_func,
    kab         : symmetric_key,
    rat, rbt,
        alice_bob_rbt,
        bob_alice_rat        : protocol_id

    intruder_knowledge={a,b,g,h}



   composition
      session(a,b,g,ack,h,kab)
   /\ session(a,i,g,ack,h,kab)
   /\ session(i,b,g,ack,h,kab)
```

```
end role

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

goal

  secrecy_of rat, rbt
  authentication_on alice_bob_rbt
  authentication_on bob_alice_rat

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
environment()
```

B.3 Results

The attack on transformed WS protocol is found in AVISPA as follow.

```
SUMMARY
  UNSAFE

DETAILS
  ATTACK_FOUND
  UNTYPED_MODEL

PROTOCOL
  /Users/trungtran/span//testsuite/results/wong_avispa4.if

GOAL
  Secrecy attack on (n14(Rbt))

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 34303 states
  Reachable  : 13966 states
  Translation: 0.02 seconds
```

```
 Computation: 2.58 seconds

ATTACK TRACE
 i -> (a,6):  start
 (a,6) -> i:  exp(g,n21(Ea))

 i -> (a,3):  start
 (a,3) -> i:  exp(g,n1(Ea))

 i -> (b,4):  g
 (b,4) -> i:  exp(g,n11(Eb)).{g.exp(g,n11(Eb)).n11(Rb).n11(Kb)}_h

 i -> (a,6):  Gb(22).Commit(22)
 (a,6) -> i:  n22(Rat).{3.{ack.n22(Rat)}_h}_kab

 i -> (b,4):  n22(Rat).{3.{ack.n22(Rat)}_h}_kab
 (b,4) -> i:  n12(Rbt).{3.n22(Rat).n12(Rbt)}_h

 i -> (a,6):  n12(Rbt).{3.n22(Rat).n12(Rbt)}_h
 (a,6) -> i:  ack.{3.{ack.n22(Rat).n12(Rbt)}_h}_kab

 i -> (b,4):  ack.{3.{ack.n22(Rat).n12(Rbt)}_h}_kab
 (b,4) -> i:  n11(Kb).n11(Rb).{4.{n11(Rb).kab}_h}_kab

 i -> (a,3):  Gb(2).{exp(g,n1(Ea)).Gb(2).n11(Rb).Kb(4)}_h
 (a,3) -> i:  n2(Rat).{3.{ack.n2(Rat)}_h}_kab

 i -> (b,10):  Ga(31)
 (b,10) -> i:  exp(g,n31(Eb)).{Ga(31).exp(g,n31(Eb)).n31(Rb).n31(Kb)}_h

 i -> (b,10):  n2(Rat).{3.{ack.n22(Rat)}_h}_kab
 (b,10) -> i:  n32(Rbt).{3.n2(Rat).n32(Rbt)}_h

 i -> (a,3):  n32(Rbt).{3.n2(Rat).n32(Rbt)}_h
 (a,3) -> i:  ack.{3.{ack.n2(Rat).n32(Rbt)}_h}_kab

 i -> (a,3):  Kb(4).n11(Rb).{4.{n11(Rb).kab}_h}_kab
 (a,3) -> i:  {a.n4(Rat)}_dummy_sk.{ack.1.a.b}_kab
              & Secret(n4(Rat),set_92);  Witness(a,b,bob_alice_rat,n4(Rat));
              & Add a to set_92;  Add b to set_92;

 i -> (b,4):  {a.n4(Rat)}_dummy_sk.{ack.1.a.b}_kab
 (b,4) -> i:  {b.n4(Rat).n14(Rbt)}_(exp(g,n11(Eb)))
```

```
              & Secret(n14(Rbt),set_110);  Witness(b,a,alice_bob_rbt,n14(Rbt));
              & Add a to set_110;  Add b to set_110;
```

Appendix C

Implementation of 2-Move Secure Device Pairing on Arduino

C.1 UDP Client Source Code

```
#include <ecc.h>
#include <string.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <sha1.h>

//client information
byte mac[] =  {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEB };
IPAddress client_ip(192,168,1,2);
unsigned int client_port = 9998;

//An EthernetUDP instance
EthernetUDP Udp;

//server information
IPAddress server_ip(192,168,1,1);
unsigned int server_port = 9999;

//client ECC key
EccPoint l_Q2;
uint8_t l_secret_server[NUM_ECC_DIGITS + 1];
uint8_t l_secret_client[NUM_ECC_DIGITS + 1];
uint8_t l_shared2[NUM_ECC_DIGITS +1];
```

```
uint8_t l_random2[NUM_ECC_DIGITS +1];
uint8_t l_shared1[NUM_ECC_DIGITS +1];// shared key from server, debugging purpose

//random of server
byte random_server[4];
//random of client
byte random_client[4];
//commitment from client
byte client_commit[20];
//decommitment from client
byte client_decommit[4];
//client say request
byte say = 0;

byte packetBuffer[UDP_TX_PACKET_MAX_SIZE +1]; //buffer to hold incoming packet,

//setup LED for visible light communication
const int ledPin = 9;      // the pin that the LED is attached to
byte byteOn = 150; //the brightness of 1
byte byteOff = 10; //the brightness of 0

unsigned long gtime1,gtime2;

//-------------------begin setup--------------------------------//
void init_key()
{
  memset(l_secret_server,0,NUM_ECC_DIGITS  + 1);
  memset(l_secret_client,0,NUM_ECC_DIGITS  + 1);
  memset(l_shared2,0,NUM_ECC_DIGITS  + 1);
  memset(l_random2,0,NUM_ECC_DIGITS  + 1);

}
void setup()
{
    //start the Ethernet UDP
   Ethernet.begin(mac,client_ip);
   Udp.begin(client_port);

   Serial.begin(9600);

   unsigned long time1,time2;
   time1 = millis();
   //generate ECC key
   randomSeed(analogRead(0));
   getRandomBytes(l_secret_client, NUM_ECC_DIGITS * sizeof(uint8_t));
   getRandomBytes(l_random2, NUM_ECC_DIGITS * sizeof(uint8_t));
   ecc_make_key(&l_Q2, l_secret_client, l_secret_client);

    time2 = millis();
    Serial.print("Key generation ms:");
    Serial.println(time2-time1);

    //generate a client random
    getRandomBytes(random_client, 4 * sizeof(uint8_t));
    printDebug("random client",random_client,4);

    calculate_commit();

    //setup for VLC
    pinMode(ledPin, OUTPUT);
   delay(500);
}
//--------------------end setup--------------------------------//

int isRequestOne = 0;

//------------------begin loop----------------------//
void loop()
{
   //send hello 99 to server until receiving the request 1 from server
   if(isRequestOne == 0)
   {
      say = 99;
      sndMsg(&say,1);
      isRequestOne =1;
   }
   //when receive a message from client
   int packetSize = Udp.parsePacket();
   if(Udp.available())
   {
     Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
    // Serial.println("Contents:");
     //printHex(packetBuffer,packetSize);
   }

   //if it is a request
   if(packetSize == 1)
   {
     //process request from client
     byte request;
     request = packetBuffer[0];
     Serial.print("Received request:");
     Serial.println(request);
     processRequestedMsg(request);
   }
   //when receive a data from client
   if(packetSize > 1)
   {
     receiveData(packetBuffer,packetSize);
   }
   if(packetSize == 0) delay(200);
}
//------------------end loop----------------------//
//send a message
void sndMsg(byte *buf, int len)
{
    Udp.beginPacket(server_ip,server_port);
    Udp.write(buf,len);
    Udp.endPacket();
    delay(300);
}
//-----------------------begin processing request---------------------------------//
//process request from server
void processRequestedMsg(int request)
{
    if(request == 1)
    {
      //send client key
      sndMsg(l_secret_client,NUM_ECC_DIGITS);
      //stop send hello to server
      isRequestOne = 1;
      //for debug
      printDebug("send client key",l_secret_client,NUM_ECC_DIGITS);
    }
    if(request == 2)
    {
      //send client commit
       sndMsg(client_commit,20);
      //for debug
      printDebug("send client commit",client_commit,20);
    }
    if(request == 21)
```

```
{
  //send request 3
  say  = 3;
  sndMsg(&say,1);
  //for debug
  Serial.println("Send request 3");
}
if(request == 5)
{
  unsigned long time1,time2;
   time1 = millis();
  //send decommit
   //padding random_server into 20byte hash key
   byte hashKey[20];
   memset(hashKey,0,20);
   memcpy(hashKey,random_server,4);
   //calculate SHA1 with key
   Sha1.initHmac(hashKey,20);
   byte hmacInput[49];
   memset(hmacInput,0,49);
   memcpy(hmacInput,l_secret_client,NUM_ECC_DIGITS);
   memcpy(hmacInput + NUM_ECC_DIGITS,l_secret_server,NUM_ECC_DIGITS);
   hmacInput[48]='\n';
   Sha1.print((char*)hmacInput);

   //take 4-frist byte of HMAC
   byte hashMac[4];
   memcpy(hashMac,Sha1.resultHmac(),4);

   //calculate decommit
   int i;
   for(i = 0;i<4;i++)
      client_decommit[i]= random_client[i]^hashMac[i];

   time2 = millis();
   Serial.print("Generating decommit value spends ms:");
   Serial.println(time2-time1);
   //send 32-bits decommit on VLC channel
   delay(1000);
   sndMsgtoLED(ledPin,client_decommit);

   printDebug("client decommit",client_decommit,4);
}
//request for debug key
```

```
    if(request == 51)
    {
      //send request 6
      say = 6;
      sndMsg(&say,1);
      Serial.println("Send request 6");
    }
}

//process received data from server
void receiveData(const byte *buf,int packetSize)
{

  if(say == 3)
  {
    //accept l_secret_server, check if the data is a key or not - based on size
     if(packetSize == NUM_ECC_DIGITS){
        memcpy(l_secret_server,buf, NUM_ECC_DIGITS);
        //for debug
        printDebug("Received server key",l_secret_server,NUM_ECC_DIGITS);

    // send say = 4
      say = 4;
      sndMsg(&say,1);
      Serial.println("Send request 4");
     }
     else
      Serial.println("Cannot parse client key");
  }
  else
  if(say == 4)
  {
    //accept server random
    if(packetSize == 4){
       memcpy(random_server,buf,4);
       //for debug
      printDebug("Received server random",random_server,4);
     }
     else
      Serial.println("Cannot parse server random value");

    //send ACK 41
    byte ACK = 41;
    sndMsg(&ACK,1);
    Serial.println("Send request 41");
  }
  //for debug only
  if(say == 6)
  {
    //generate shared key

     if(packetSize == 2)
     {
       if(buf[0] == 1 && buf[1] == 1)
         {
           Serial.println("Server accepts the connection");
            generate_sharedkey();
         }
       else if(buf[0] == 0 && buf[1] == 0)
         {
            Serial.println("Server rejects the connection");
         }
     }
  }
}

//----------------------tools------------------------//
void getRandomBytes(uint8_t *arr,int arrLen)
{
  int i;
  for(i =0;i<arrLen;i++)
    arr[i]=random(0,256);
}

void printHex(const byte* arr,int len)
{
  int i;
  char ptr1[10];
  char ptr2[10];
  String mystring;
  for (i=0; i<len; i++) {
      sprintf(ptr1,"%x",arr[i]>>4);
      sprintf(ptr2,"%x",arr[i]&0xf);
      mystring += ptr1;
      mystring += ptr2;
      Serial.print(mystring);
      mystring.remove(0,mystring.length());
  }
  Serial.println("");
}

//generate a shared key
void generate_sharedkey(){
       //generate a shared key
   unsigned long time1,time2;
   time1 = millis();
   if (!ecdh_shared_secret(l_shared2, &l_Q2, l_secret_server,l_random2))
    {
        Serial.println("shared_secret() failed (2)\n");
           return ;
    }
    time2 = millis();
    Serial.print("Generating shared key spends ms:");
    Serial.println(time2-time1);
    Serial.print("shared_secret:");
    printHex(l_shared2,NUM_ECC_DIGITS);
}

void printDebug(char *text,const byte *arr, int len)
{
   Serial.println(text);
   printHex(arr,len);
}

//----------------------send LED to server----------------------
void sndMsgtoLED(int LedPin,byte *rnd)
{
  Serial.println("Send message via LED");
   unsigned long time1,time2;
   time1 = millis();
  int i,j;
  byte temp[4];
  memset(temp,0,4);

  byte isOn =0;
  for(j =0;j<4;j++)
  {
    temp[j] =1;
    for(i=0;i<8;i++)
    {
        isOn = temp[j] & rnd[j];
        if(isOn > 0)
        {
          analogWrite(LedPin, byteOn);
          Serial.print("1");
          delay(100);
        }
        else
        {
          analogWrite(LedPin, byteOff);
          Serial.print("0");
          delay(100);
        }
        temp[j] = temp[j] << 1;
    }
    Serial.print(" ");
  }
  Serial.println("");
  analogWrite(LedPin, 0);
  time2 = millis();
  Serial.print("Sending 32-bits via VLC spends ms:");
  Serial.println(time2-time1);
}

void calculate_commit()
{
   unsigned long time1,time2;
   time1 = millis();
   unsigned char hashKey = 0;
   Sha1.init();
   Sha1.initHmac(&hashKey,1);
   char input[29];
   memset(input,0,29);
   memcpy(input,l_secret_client,24);
   input[24] = random_client[0];
   input[25] = random_client[1];
   input[26] = random_client[2];
   input[27] = random_client[3];
   Sha1.print(input);
   memcpy(client_commit,Sha1.resultHmac(),20);
   time2 = millis();
   Serial.print("Calculating commitment spends ms:");
   Serial.println(time2-time1);
}
```

C.2 UDP Server Source Code

```
#include <ecc.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <string.h>
#include <sha1.h>

//EEC Keys
EccPoint l_Q1;
uint8_t l_secret_server[NUM_ECC_DIGITS  + 1];
uint8_t l_secret_client[NUM_ECC_DIGITS + 1];
uint8_t l_shared1[NUM_ECC_DIGITS + 1];
uint8_t l_random1[NUM_ECC_DIGITS +1 ];

//server information
byte mac[] =  {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEA };
IPAddress server_ip(192,168,1,1);
unsigned int server_port = 9999;

//An EthernetUDP instance
EthernetUDP Udp;

//random of server
byte random_server[4];
//random of client
byte random_client[4];
//commitment from client
byte client_commit[20];
//decommitment from client
byte client_decommit[4];

byte ACK = 0;

byte packetBuffer[UDP_TX_PACKET_MAX_SIZE + 1]; //buffer to hold incoming packet,

//visible light communication
int photocellPin = 0;     // the cell and 10K pulldown are connected to a0
int brightZero = 600;
int brightOne = 900;
int  photocellReading; //photocell value
byte vlcBuffer[4];
byte isAccept = 0;

//-------------------begin setup--------------------------------//
void init_key()
{
  memset(l_secret_server,0,NUM_ECC_DIGITS  + 1);
  memset(l_secret_client,0,NUM_ECC_DIGITS  + 1);
  memset(l_shared1,0,NUM_ECC_DIGITS  + 1);
  memset(l_random1,0,NUM_ECC_DIGITS  + 1);
}
void setup()
{
   //start the Ethernet UDP
   Ethernet.begin(mac,server_ip);
   Udp.begin(server_port);
   Serial.begin(9600);
   init_key();
   //generate ECC random and key
   randomSeed(analogRead(0));
   getRandomBytes(l_secret_server, NUM_ECC_DIGITS * sizeof(uint8_t));
   getRandomBytes(l_random1, NUM_ECC_DIGITS * sizeof(uint8_t));
   ecc_make_key(&l_Q1, l_secret_server, l_secret_server);

   //generate a random RA
    getRandomBytes(random_server, 4 * sizeof(uint8_t));
    //setup for VLC
    memset(vlcBuffer,0,4);
    delay(500);
}
//-------------------end setup--------------------------------//
//------------------------------------------------------------//
byte say = 0;

//send a message
void sndMsg(byte *buf, int len)
{
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write(buf,len);
    Udp.endPacket();
    delay(300);
}

//process request from client
void processRequestedMsg(int request)
```

```
{
    if(request == 99)
    {
      //send say = 1
      if(say == 0)
      {
        say =1;
        Serial.println("Send ACK = 1");
        sndMsg(&say,1);
        isAccept = 0;
      }
    }
    if(request == 3)
    {
      //send l_secret_server
      if(say == 21)
      {
         Serial.println("Send server key");
         sndMsg(l_secret_server,NUM_ECC_DIGITS);
      }
    }
    if(request == 4)
    {
      //send ra
        Serial.println("Send server's random ");
        sndMsg(random_server,4);
    }
    if(request == 41)
    {
      //send say = 5
         say = 5;
         sndMsg(&say,1);
         photoReading();
         //---received client decommit then extract ra
         memcpy(client_decommit,vlcBuffer,4);
         printDebug("Received client decommit",client_decommit,4);
          extract_client_random();

          //send ACK 51
          ACK = 51;
          Serial.println("Send ACK = 51");
          sndMsg(&ACK,1);
          //generate random key when commitment is checked
          //commitment_check();
          if(commitment_check() == 0)
          {
             generate_sharedkey();
             isAccept = 1;
          }
          else
            isAccept = 0;
     }
    //request for debug shared key
    if(request == 6)
    {
       byte OK[2];
       if(isAccept == 1)
       {
         OK[0] =1;
         OK[1] = 1;
         sndMsg(OK,2);
       }
       else
       {
         OK[0] = 0;
         OK[1] = 0;
         sndMsg(OK,2);
       }
      say =0;
    }
}

//process received data from client
void receiveData(const byte *buf, int packetSize)
{
  if(say == 1)
  {
    //accept l_secret_client, check if the data is a key or not - based on size
     if(packetSize == NUM_ECC_DIGITS){

        memcpy(l_secret_client,buf,NUM_ECC_DIGITS);
        printDebug("Received client key",l_secret_client,NUM_ECC_DIGITS);
            // send say = 2
        say = 2;
        sndMsg(&say,1);
        Serial.println("Send request 2");
     }
     else
```

```
      Serial.println("Cannot parse client key");
  }
  else
  if(say == 2)
  {
    //accept commit h(l_client_secret,random_client)
    if(packetSize == 20){
       memcpy(client_commit,buf,20);
       printDebug("Received client commit",client_commit,20);
      //send ACK 21
       say = 21;
       sndMsg(&say,1);
       Serial.println("Send request 21");
     }
     else
      Serial.println("Cannot parse client commit value");
  }
}

//------------------begin loop----------------------//
void loop()
{
   int packetSize = Udp.parsePacket();

   if(Udp.available())
   {
     Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
     //Serial.println("Contents:");
     //printHex(packetBuffer,packetSize);
   }

   //when receive a request from client

   if(packetSize == 1)
   {
     //process request from client
     byte request;
     request = packetBuffer[0] ;
     Serial.print("Received request:");
     Serial.println(request);
     processRequestedMsg(request);
   }
   //when receive a data from client
   if(packetSize > 1)
   {
     receiveData(packetBuffer,packetSize);
   }
   if(packetSize == 0) delay(200);
}
//------------------end loop-----------------------//
//----------------------tools-------------------------//
void getRandomBytes(uint8_t *arr,int arrLen)
{
  int i;
  for(i =0;i<arrLen;i++)
    arr[i]=random(0,256);
}

void printHex(const byte* arr,int len)
{
  int i;
  char ptr1[10];
  char ptr2[10];
  String mystring;
  for (i=0; i<len; i++) {
      sprintf(ptr1,"%x",arr[i]>>4);
      sprintf(ptr2,"%x",arr[i]&0xf);
      mystring += ptr1;
      mystring += ptr2;
      Serial.print(mystring);
      mystring.remove(0,mystring.length());
  }
  Serial.println("");
}

//generate a shared key
void generate_sharedkey(){
       //generate a shared key
   if (!ecdh_shared_secret(l_shared1, &l_Q1, l_secret_client,l_random1))
    {
        Serial.println("shared_secret() failed (2)\n");
           return ;
    }
    Serial.print("shared_secret:");
    printHex(l_shared1,NUM_ECC_DIGITS);
}
void printDebug(char *text,const byte *arr, int len)
{
   Serial.println(text);
   printHex(arr,len);
}

//----------------receive decommit value on VLC channel
void photoReading()
{
   int loop_count =0;
   int index = 0;
   memset(vlcBuffer,0,4);
   while(loop_count <1000)
   {
       photocellReading = analogRead(photocellPin);
       if(photocellReading < brightZero)
       {
          //Serial.println(" - ");
          //index = 0;
          delay(10);
       }
      else{
          delay(45);
          for(index = 1;index <=32;index ++)
          {
             if(photocellReading < brightZero)
             {
                 delay(5);
                 index = index -1;
             }
             if(photocellReading <= brightOne && photocellReading > brightZero)
             {
                 //if(index <= 32)
                 //   VLCBuffer[index-1] = 0;
                 delay(100);
             }else
            if(photocellReading >= brightOne)
            {
                 //Serial.print("1");
                 if(index <= 32){
                  vlcBuffer[(index-1)/8] = vlcBuffer[(index-1)/8] | ((byte)1<<((index-1)%8));
                 delay(98);
                 }
             }
             photocellReading = analogRead(photocellPin);
          }
```

```
           //print_arr();
           break;
      }
      loop_count++;
   }
}

void extract_client_random()
{
       //calculate hmac
       //padding random_server into 20byte hash key
       byte hashKey[20];
       memset(hashKey,0,20);
       memcpy(hashKey,random_server,4);
       //calculate SHA1 with key
       Sha1.initHmac(hashKey,20);
       byte hmacInput[49];
       memset(hmacInput,0,49);
       memcpy(hmacInput,l_secret_client,NUM_ECC_DIGITS);
       memcpy(hmacInput + NUM_ECC_DIGITS,l_secret_server,NUM_ECC_DIGITS);
       hmacInput[48]='\n';
       Sha1.print((char*)hmacInput);

       //take 4-frist byte of HMAC
       byte hashMac[4];
       memcpy(hashMac,Sha1.resultHmac(),4);
       //calculate random_client by xor client_decommit with hashMac
       int i;
       for(i = 0;i<4;i++)
        random_client[i]= client_decommit[i]^hashMac[i];
       //print debug
       printDebug("extracting random_client:",random_client,4);
}

int commitment_check()
{
    unsigned char hashKey = 0;
    char hashResult[20];
    Sha1.init();
    Sha1.initHmac(&hashKey,1);

   char input[29];
   memset(input,0,29);
   memcpy(input,l_secret_client,24);
   input[24] = random_client[0];
   input[25] = random_client[1];
   input[26] = random_client[2];
   input[27] = random_client[3];
   Sha1.print(input);
   memcpy(hashResult,Sha1.resultHmac(),20);
   printDebug("hashResult:",(byte*)hashResult,20);
    if(memcmp(hashResult,client_commit,20) == 0)
    {
        Serial.println("commitment checked successfully");
        return 0;
    }
    else
      Serial.println("commitment checked fail");
      return 1;
}
```

Bibliography

[1] UeliM. Maurer and PierreE. Schmid. A calculus for secure channel establishment in open networks. In Dieter Gollmann, editor, *Computer Security — ESORICS 94*, volume 875 of *Lecture Notes in Computer Science*, pages 173–192. Springer Berlin Heidelberg, 1994. ISBN 978-3-540-58618-0. doi: 10.1007/3-540-58618-0_63. URL http://dx.doi.org/10.1007/3-540-58618-0_63.

[2] F.Javier Thayer, Vipin Swarup, and JoshuaD. Guttman. Metric strand spaces for locale authentication protocols. In Masakatsu Nishigaki, Audun JAzsang, Yuko Murayama, and Stephen Marsh, editors, *Trust Management IV*, volume 321 of *IFIP Advances in Information and Communication Technology*, pages 79–94. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13445-6. doi: 10.1007/978-3-642-13446-3_6. URL http://dx.doi.org/10.1007/978-3-642-13446-3_6.

[3] Khan Pathan Al-Sakib. *Security of Self-Organizing Networks: MANET, WSN, WMN, VANET*. ISBN: 978-1-4398-1920-3. Auerbach Publications, October 14, 2010.

[4] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In Bruce Christianson, Bruno Crispo, JamesA. Malcolm, and Michael Roe, editors, *Security Protocols*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–182. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-67381-1. doi: 10.1007/10720107_24. URL http://dx.doi.org/10.1007/10720107_24.

[5] M.T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun. Loud and clear: Human-verifiable authentication based on audio. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 10–10, 2006. doi: 10.1109/ICDCS.2006.52.

[6] Claudio Soriente, Gene Tsudik, and Ersin Uzun. Hapadep: Human-assisted pure audio device pairing. In *Proceedings of the 11th international conference on Information Security*, ISC '08, pages 385–400, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85884-3. doi: 10.1007/978-3-540-85886-7_27. URL http://dx.doi.org/10.1007/978-3-540-85886-7_27.

[7] Felix Xiaozhu Lin, Daniel Ashbrook, and Sean White. Rhythmlink: securely pairing i/o-constrained devices by tapping. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 263–272, New York, NY, USA,

2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047231. URL http://doi.acm.org/10.1145/2047196.2047231.

[8] Nitesh Saxena, Md. Borhan Uddin, and Jonathan Voris. Universal device pairing using an auxiliary device. In *Proceedings of the 4th symposium on Usable privacy and security*, SOUPS '08, pages 56–67, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-276-4. doi: 10.1145/1408664.1408672. URL http://doi.acm.org/10.1145/1408664.1408672.

[9] Young Sam Kim, Seung-Hyun Kim, and Seung-Hun Jin. Srs-based automatic secure device pairing on audio channels. In *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*, pages 1–6, 2010.

[10] Stephan Sigg, Dominik Schuermann, and Yusheng Ji. Pintext: A framework for secure communication based on context. In Alessandro Puiatti and Tao Gu, editors, *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, volume 104 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 314–325. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30972-4. doi: 10.1007/978-3-642-30973-1_31. URL http://dx.doi.org/10.1007/978-3-642-30973-1_31.

[11] Michael T. Goodrich, Michael Sirivianos, John Solis, Claudio Soriente, Gene Tsudik, and Ersin Uzun. Using audio in secure device pairing. *Int. J. Secur. Netw.*, 4(1/2):57–68, February 2009. ISSN 1747-8405. doi: 10.1504/IJSN.2009.023426. URL http://dx.doi.org/10.1504/IJSN.2009.023426.

[12] J.M. McCune, A. Perrig, and M.K. Reiter. Seeing-is-believing: using camera phones for human-verifiable authentication. In *Security and Privacy, 2005 IEEE Symposium on*, pages 110–124, 2005. doi: 10.1109/SP.2005.19.

[13] Adrian Perrig and Dawn Song. Hash visualization: a new technique to improve real-world security. In *In International Workshop on Cryptographic Techniques and E-Commerce*, pages 131–138, 1999.

[14] Carl Ellison and Steve Dohrmann. Public-key support for group collaboration. *ACM Trans. Inf. Syst. Secur.*, 6(4):547–565, November 2003. ISSN 1094-9224. doi: 10.1145/950191.950195. URL http://doi.acm.org/10.1145/950191.950195.

[15] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6 pp.–313, 2006. doi: 10.1109/SP.2006.35.

[16] Claudio Soriente, Gene Tsudik, and Ersin Uzun. Beda: Button-enabled device association, 2007.

[17] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel: Design and usability study. *Information Forensics and Security, IEEE Transactions on*, 6(1):28–38, 2011. ISSN 1556-6013. doi: 10.1109/TIFS.2010.2096217.

[18] Rene Mayrhofer, Mike Hazas, and Hans Gellersen. An authentication protocol using ultrasonic ranging. Technical report, 2006.

[19] GeorgeT. Amariucai, Clifford Bergman, and Yong Guan. An automatic, time-based, secure pairing protocol for passive rfid. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy*, volume 7055 of *Lecture Notes in Computer Science*, pages 108–126. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-25285-3. doi: 10.1007/978-3-642-25286-0_8. URL http://dx.doi.org/10.1007/978-3-642-25286-0_8.

[20] R. Mayrhofer and M. Welch. A human-verifiable authentication protocol using visible laser light. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 1143–1148, 2007. doi: 10.1109/ARES.2007.5.

[21] Ileana Buhan, Jeroen Doumen, Pieter Hartel, and Raymond Veldhuis. Feeling is believing: a location limited channel based on grip pattern biometrics and cryptanalysis.

[22] LarsErik Holmquist, Friedemann Mattern, Bernt Schiele, Petteri Alahuhta, Michael Beigl5, and Hans-W. Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In GregoryD. Abowd, Barry Brumitt, and Steven Shafer, editors, *Ubicomp 2001: Ubiquitous Computing*, volume 2201 of *Lecture Notes in Computer Science*, pages 116–122. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-42614-1. doi: 10.1007/3-540-45427-6_10. URL http://dx.doi.org/10.1007/3-540-45427-6_10.

[23] Jonathan Lester, Blake Hannaford, and Gaetano Borriello. Are you with me?" – using accelerometers to determine if two devices are carried by the same person. In *In Proceedings of Second International Conference on Pervasive Computing (Pervasive 2004*, pages 33–50, 2004.

[24] Rene Mayrhofer and Hans Gellersen. Shake well before use: Authentication based on accelerometer data. In Anthony LaMarca, Marc Langheinrich, and KhaiN. Truong, editors, *Pervasive Computing*, volume 4480 of *Lecture Notes in Computer Science*, pages 144–161. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-72036-2. doi: 10.1007/978-3-540-72037-9_9. URL http://dx.doi.org/10.1007/978-3-540-72037-9_9.

[25] Ahren Studer, Timothy Passaro, and Lujo Bauer. Don't bump, shake on it: the exploitation of a popular accelerometer-based smart phone exchange and its secure replacement. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 333–342, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0672-0. doi: 10.1145/2076732.2076780. URL http://doi.acm.org/10.1145/2076732.2076780.

[26] Bogdan Groza and Rene Mayrhofer. Saphe: simple accelerometer based wireless pairing with heuristic trees. In *Proceedings of the 10th International Conference on Advances in Mobile Computing &#38; Multimedia*, MoMM '12, pages 161–168, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1307-0. doi: 10.1145/2428955.2428989. URL http://doi.acm.org/10.1145/2428955.2428989.

[27] Ming Ki Chong, Gary Marsden, and Hans Gellersen. Gesturepin: using discrete gestures for associating mobile devices. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, pages 261–264, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-835-3. doi: 10.1145/1851600.1851644. URL http://doi.acm.org/10.1145/1851600.1851644.

[28] Oyuntungalag Chagnaadorj and Jiro Tanaka. Mimicgesture: Secure device pairing with accelerometer-based gesture input. In Youn-Hee Han, Doo-Soon Park, Weijia Jia, and Sang-Soo Yeo, editors, *Ubiquitous Information Technologies and Applications*, volume 214 of *Lecture Notes in Electrical Engineering*, pages 59–67. Springer Netherlands, 2013. ISBN 978-94-007-5856-8. doi: 10.1007/978-94-007-5857-5_7. URL http://dx.doi.org/10.1007/978-94-007-5857-5_7.

[29] Claude Castelluccia and Pars Mutaf. Shake them up!: a movement-based pairing protocol for cpu-constrained devices. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05, pages 51–64, New York, NY, USA, 2005. ACM. ISBN 1-931971-31-5. doi: 10.1145/1067170.1067177. URL http://doi.acm.org/10.1145/1067170.1067177.

[30] Adin Scannell, Alexander Varshavsky, and Anthony Lamarca. Amigo: Proximity-based authentication of mobile devices. In *In Ubicomp*, pages 253–270, 2007.

[31] S. Mirzadeh, H. Cruickshank, and R. Tafazolli. Secure device pairing: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):17–40, First 2014. ISSN 1553-877X. doi: 10.1109/SURV.2013.111413.00196.

[32] Secure communication method and apparatus. U.S. Patent Number 5,450,493, sep 1995.

[33] Dirk Balfanz Smetters, Dirk Balfanz, D. K. Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. 2002.

[34] C. Mitchell, C. Gehrmann, and K. Nyberg. Manual authentication for wireless devices. *Cryptobytes*, January 2004. URL http://eprints.rhul.ac.uk/562/.

[35] K. Nyberg. C. Gehrmann. Security in personal area networks. pages 191–230. IEE, In C. J. Mitchell, 2004.

[36] Jaap-Henk Hoepman. The ephemeral pairing problem. In Ari Juels, editor, *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 212–226. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22420-4. doi: 10.1007/978-3-540-27809-2_22. URL http://dx.doi.org/10.1007/978-3-540-27809-2_22.

[37] L. H. Nguyen and A. W. Roscoe. Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey, 2009.

[38] Ford Long Wong and Frank Stajano. Multichannel security protocols. *IEEE Pervasive Computing*, 6(4):31–39, 2007. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2007.76.

[39] Serge Vaudenay. Secure communications over insecure channels based on short authenticated strings. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 309–326. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28114-6. doi: 10.1007/11535218_19. URL http://dx.doi.org/10.1007/11535218_19.

[40] Sylvain Pasini and Serge Vaudenay. Sas-based authenticated key agreement. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 395–409. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-33851-2. doi: 10.1007/11745853_26. URL http://dx.doi.org/10.1007/11745853_26.

[41] Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In David Pointcheval, Yi Mu, and Kefei Chen, editors, *Cryptology and Network Security*, volume 4301 of *Lecture Notes in Computer Science*, pages 90–107. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-49462-1. doi: 10.1007/11935070_6. URL http://dx.doi.org/10.1007/11935070_6.

[42] M. Cagalj, S. Capkun, and J-P Hubaux. Key agreement in peer-to-peer wireless networks. *Proceedings of the IEEE*, 94(2):467–478, 2006. ISSN 0018-9219. doi: 10.1109/JPROC.2005.862475.

[43] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In DouglasR. Stinson, editor, *Advances in Cryptology — CRYPTO' 93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer Berlin Heidelberg, 1994. ISBN 978-3-540-57766-9. doi: 10.1007/3-540-48329-2_21. URL http://dx.doi.org/10.1007/3-540-48329-2_21.

[44] Frank Stajano, Graeme Jenkinson, Jeunese Payne, Max Spencer, Quentin Stafford-Fraser, and Chris Warrington. Bootstrapping adoption of the pico password replacement system. In Bruce Christianson, James Malcolm, Vashek MatyÃ!'Å!', Petr Å venda, Frank Stajano, and Jonathan Anderson, editors, *Security Protocols XXII*, volume 8809 of *Lecture Notes in Computer Science*, pages 172–186. Springer International Publishing, 2014. ISBN 978-3-319-12399-8. doi: 10.1007/978-3-319-12400-1_17. URL http://dx.doi.org/10.1007/978-3-319-12400-1_17.

[45] F.J. Thayer Fabrega, J.C. Herzog, and J.D. Guttman. Strand spaces: why is a security protocol correct? In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 160–171, May 1998. doi: 10.1109/SECPRI.1998.674832.

[46] J. Thayer, J. Herzog, and J. Guttman. Honest ideals on strand spaces. In *Proceedings of the 11th IEEE Workshop on Computer Security Foundations*, CSFW '98, pages 66–, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8488-7. URL http://dl.acm.org/citation.cfm?id=794198.795096.

[47] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *Proceedings of the 12th IEEE Workshop on Computer Security Foundations*, CSFW '99, pages 72–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0201-6. URL http://dl.acm.org/citation.cfm?id=794199.795113.

[48] Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theor. Comput. Sci.*, 283(2):333–380, June 2002. ISSN 0304-3975. doi: 10.1016/S0304-3975(01)00139-6. URL http://dx.doi.org/10.1016/S0304-3975(01)00139-6.

[49] Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Skeletons, homomorphisms, and shapes: Characterizing protocol executions. *Electron. Notes Theor. Comput. Sci.*, 173:85–102, April 2007. ISSN 1571-0661. doi: 10.1016/j.entcs.2007.02.029. URL `http://dx.doi.org/10.1016/j.entcs.2007.02.029`.

[50] Allaa Kamil and Gavin Lowe. Analysing tls in the strand spaces model. *J. Comput. Secur.*, 19(5):975–1025, September 2011. ISSN 0926-227X. URL `http://dl.acm.org/citation.cfm?id=2590701.2590707`.

[51] J.C. Herzog. The diffie-hellman key-agreement scheme in the strand-space model. In *Computer Security Foundations Workshop, 2003. Proceedings. 16th IEEE*, pages 234–247, June 2003. doi: 10.1109/CSFW.2003.1212716.

[52] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978. ISSN 0001-0782. doi: 10.1145/359657.359659. URL `http://doi.acm.org/10.1145/359657.359659`.

[53] Joshua D. Guttman and F. Javier Thayer. Authentication tests. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 96–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0665-8. URL `http://dl.acm.org/citation.cfm?id=882494.884403`.

[54] D. Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, Mar 1983. ISSN 0018-9448. doi: 10.1109/TIT.1983.1056650.