



Master Thesis

Security properties of device pairing protocols

Author(s):

Ischi, Reto

Publication Date:

2009

Permanent Link:

<https://doi.org/10.3929/ethz-a-005772956> →

Rights / License:

[In Copyright - Non-Commercial Use Permitted](#) →

This page was generated automatically upon download from the [ETH Zurich Research Collection](#). For more information please consult the [Terms of use](#).

Master Thesis

Security Properties of Device Pairing Protocols

Reto Ischi

Supervisor: Prof. Dr. David Basin
Assistant: Patrick Schaller

ETH Zurich
Information Security Department

March 9, 2009

Acknowledgment

I would like to thank my supervisor, Prof. Dr. David Basis, for his commitment to this project. I am deeply grateful to Patrick Schaller, for the time he spent on my behalf, his guidance and feedback. Finally, I thank my family and all my friends for their support and encouragement throughout this thesis.

Affidavit

I hereby declare that this master thesis has been written only by the undersigned and without any assistance from third parties. Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

This thesis has not yet been presented to any examination authority, neither in this form nor in a modified version.

(Place), (Date)

(Signature)

Abstract

In this thesis I use the term *security relation* to refer to a certain distribution of cryptographic keys between two devices. This may be the knowledge of a secret symmetric key or the possession of an authentic public key. A security relation is typically needed by a cryptographic protocol to reach a certain security goal such as authenticity, privacy or integrity of data to be transmitted. A *device pairing protocol* allows us to create such a security relation.

Device pairing protocols profit from the presence of the user and physical properties of communication channels, to create a security relation. Depending on the protocol the user examines certain properties or accomplishes certain tasks. A successful protocol run then guarantees that the security relation was created between the two chosen devices.

Diffie-Hellman key agreement seems to be suitable to create a shared secret key between two devices, because the protocol is secure against a passive attacker. To defend an active attacker, the exchanged messages have to be authenticated. A number of device pairing protocols are proposed to do this. Other protocols that do not rely on public key techniques create a shared secret key by having exclusive access to random data, generated or provided by the user.

Although the integration of the user into the protocols can be very different, the functional principles of the protocols are often quite similar. This allows to divide the protocols into different classes, which can further be described by transformations of security primitives. This abstraction allows a better understanding of current protocols and to identify the necessary requirements for further ones.

Zusammenfassung

Unter einer *Sicherheitsrelation* zwischen zwei Geräten verstehe ich in dieser Arbeit eine bestimmte Verteilung von kryptographischen Schlüsseln. Dies kann zum Beispiel die Kenntnis eines geheimen, symmetrischen Schlüssels oder der Besitz eines authentischen öffentlichen Schlüssels sein. Eine solche Sicherheitsrelation wird typischerweise von einem kryptographischen Protokoll zur Erreichung eines bestimmten Sicherheitsziels wie Authentizität, Vertraulichkeit oder Integrität von zu übertragenden Daten benötigt. Ein *Device-Pairing Protokoll* erlaubt einem Benutzer eine solche Sicherheitsrelation zwischen zwei Geräten zu erstellen.

Zur Erzeugung einer Sicherheitsrelation nutzen die Protokolle die Präsenz des Benutzers sowie physikalische Eigenschaften der Kommunikationskanäle. In Abhängig des Protokolls überprüft der Benutzer gewisse Bedingungen oder führt bestimmte Aufgaben durch. Ein erfolgreicher Protokolldurchlauf garantiert dem Benutzer, dass eine Sicherheitsrelation genau zwischen den Geräte zustande gekommen ist, welche der Benutzer zuvor ausgewählt hat.

Die Diffie-Hellman Schlüsselvereinbarung scheint zur Erzeugung eines gemeinsamen, geheimen Schlüssels geeignet zu sein, da das Protokoll vor einem passiven Angreifer schützt. Um einen aktiven Angriff auszuschliessen, müssen die ausgetauschten Nachrichten authentisiert werden, wofür eine Reihe verschiedener Device-Pairing Protokolle vorgeschlagen wurden. Andere Protokolle, welche ohne Public-Key-Techniken auskommen, erzeugen einen geheimen, gemeinsamen Schlüssel durch exklusiven Zugriff auf zufällige Daten, welche durch den Benutzer generiert oder zur Verfügung gestellt werden.

Obwohl die Einbindung des Benutzers in das Device-Pairing Protokoll sehr unterschiedlich sein kann, so sind die Funktionsprinzipien der Protokolle vielfach ähnlich. Dies ermöglicht es, die Protokolle in Klassen zu unterteilen, welche ferner durch Transformationen von Sicherheitsprimitiven beschrieben werden können. Diese Abstraktion erlaubt es bestehende Protokolle besser zu verstehen und die notwendigen Bedingungen für zukünftige Device-Pairing-Protokoll aufzuzeigen.

Contents

1	Introduction	9
1.1	Document Structure	9
1.2	Previous and Related Work	9
1.3	Motivation Example	10
1.4	Terms and Device Setup	11
1.5	Security Relation	13
1.6	Definition of Device Pairing	14
1.7	Trade-Off Between Security and Usability	14
2	Use Cases	17
3	Device Pairing Protocols	21
3.1	Notations	22
3.2	General Protocol Assumptions	23
3.3	General Attacker Model	23
3.4	One-Way Public Key Authentication	23
3.5	OOB Transfer of Message Digests (P1)	24
3.6	Shared Secret Key Generation (P2-P8)	27
3.7	OOB Key Transfer Protocols (P2)	27
3.8	Key Agreement Protocols (P3-P8)	27
3.9	User-Assisted Key Agreement (P3)	28
3.10	OOB Key Exchange (P4)	31
3.11	Diffie-Hellman Key Agreement (P5-P8)	33
3.12	Both-Way Public Key Authentication	34
3.13	Authentication by Integrity Checksum (P5,P6)	35
3.14	Authentication by Integrity Checksum - User-Assisted (P5)	35
3.15	Authentication by Integrity Checksum - OOB Transfer (P6)	39
3.16	Authentication Using a Shared Secret (P7,P8)	43
3.17	Authentication using a Shared Secret - User-Assisted (P7)	43
3.18	Another Protocol Classification	47
4	Attacks	49
4.1	Compromising Devices	49
4.2	Denial-of-Service Attacks	49
4.3	Man-in-the-Middle Attacks	50
4.4	Bluetooth Pairing Attack	52

4.5	Reflection Attack	52
4.6	Attacks against Distance-Bounding Protocols	52
4.7	Another Attacks Against Shake Them Up!	54
5	Security Properties	55
5.1	Terms and Notations	56
5.2	OOB Transfer of Key Commitments (P1)	58
5.3	OOB Key Transfer (P2)	58
5.4	User-Assisted Key Agreement (P3)	59
5.5	OOB Key Exchange (P4)	59
5.6	Diffie-Hellman Key Agreement (P5-P8)	60
5.7	User-Assisted Key Agreement (P5)	61
5.8	Authentication by Integrity Checksum - OOB Transfer (P6)	62
5.9	Authentication using a Shared Secret Key (P7,P8)	65
5.10	Authentication using a Shared Secret Key - User-Assisted (P7)	66
	Conclusions	69

Security is always excessive until it's not enough.

(Robbie Sinclair)

1 Introduction

Electronic devices get more and more important in our daily life since we use them for even more tasks. For many of these tasks two devices need to communicate with each other, so that no unauthorized person is able to affect or eavesdrop the communication. This can either be done by using a physically secure communication medium or by using an insecure communication medium together with a suitable cryptographic protocol. These protocols typically rely on some information, shared between the communicating devices (e.g. a secret key). If this information is missing we have to create it. This is done with device pairing protocols.

Device pairing protocols can mainly be found in ubiquitous computing environments, where users want to establish secure ad-hoc communications over wireless standards like Wi-Fi (IEEE 802.11) or Bluetooth. This is because the underlying communication medium is insecure and the devices often do not know each other in advance and therefore do not share any information useful for a cryptographic protocol.

This thesis is a comparative study of device pairing protocols, with focus on the security properties of these protocols. I present a selection of different protocols, compare them and point out what they indeed achieve and how they achieve that. I further show, how such protocols can be attacked and how they should be designed to prevent attacks.

1.1 Document Structure

Chapter 1 introduces the topic, describes the used terms and the device setup. In Chapter 2, I present possible application scenarios. Chapter 3 describes the attacker model, gives an overview of today's most important protocols and classifies them. In Chapter 4, I present a selection of attacks against pairing protocols. Finally, Chapter 5 describes the security properties of device pairing protocols by transformations of security primitives.

1.2 Previous and Related Work

There is a large body of relevant prior work on the general topic of device pairing. Stajano and Anderson [22] propose the *resurrecting duckling* security policy model, where keys are established by physical contact of the communication parties. Balfanz, *et al.* [1] made progress by suggesting the use of infrared communication as a human verifiable side-channel.

Other proposed side-channels are: visual channels [15, 18], audio channels [10, 21] and secured radio channels [5, 6]. Mayrhofer and Gellersen propose to use the context of a device for sender authentication [14]. A location based protocol is proposed by Čagalj, Čapkun and Hubaux [4]. All these approaches are described in more detail in Chapter 3. I further acknowledge the work of Suomalainen, Valkonen and Asokan for their comparative analysis and classification of key agreement protocols [23].

1.3 Motivation Example

Consider a user, who wants to print a business contract, just received by e-mail on his handheld, in a public place (e.g. at an airport). There are several printers he can choose from. His goal is to print the document with the printer in front of him and the document should be kept secret, i.e., no-one else should be able to read its content. The user assumes that the printer is “trusted”, meaning that it is not compromised by an unauthorized person. How could this be realized?

Idea 1 The user connects his handheld to the printer using a printer cable and prints the document.

Problem There is no printer cable available to connect the two devices or the handheld has no appropriate interface, and of course, the user does not want to carry a printer cable with him all the time.

Conclusion We want to use wireless communication (e.g. Wi-Fi IEEE 802.11) to print the document.

Idea 2 The user scans the WLAN for all available printers, chooses the right one and sends the document to the printer.

Problem An adversary nearby could overhear the communication and read the document’s content.

Conclusion The user needs to send the document in an encrypted form to the printer.

To encrypt the document, the handheld needs a key. If we use symmetric cryptography, the printer has to possess the same key to decrypt the document. Unfortunately, the handheld and the printer do not share a key.

Idea 3 The printer or handheld generates a symmetric key and sends it over the WLAN to the other device.

Problem The key can be intercepted by the adversary and becomes useless.

Conclusion Symmetric keys must not be transferred in plaintext over an insecure network.

Idea 4 Why not use public key cryptography to encrypt the document? In this case, an

adversary has no benefit if he intercepts the public key. The user now asks the printer to send his public key over the WLAN to his handheld. The handheld can then encrypt the document with the printer's public key and send the cyphertext to the printer.

Problem As the adversary controls the WLAN, he can replace the printer's public key by his own public key (*man-in-the-middle* attack).

Conclusion The handheld needs an authentic copy of the printer's public key.

Idea 5 The printer indicates his public key on its display and the user transcribes the key using the handheld's keyboard.

Problem This would solve our problem, but unfortunately this procedure is not very user-friendly because no-one wants to transcribe a several hundred bit long key using a uncomfortable small handheld keyboard.

Idea 5 is an example of a *device pairing protocol* used to transfer public keys between two devices.

The handheld has now an authentic copy of the printer's public key, can encrypt the document with this key and send it over the insecure WLAN to the printer. The printer prints the document and the user leaves the public place together with the printed business contract. If the printing protocol is designed poorly, an adversary who overhears the communication could send the same encrypted printing job to the printer, which prints the same document again and the adversary gets a printed version of the document (replay attack).

This attack points out that the device pairing protocol, executed in Idea 5, can not ensure that the following protocols are secure. Since this thesis is about device pairing protocols, I do not further consider the security of protocols following a device pairing protocol.

1.4 Terms and Device Setup

The term *device* relates to a contrivance, serving one or more particular purposes, such as a computer mouse or a mobile phone. I require here that a device has at least a microprocessor, a memory and one or more input and output interfaces. Examples of device input interfaces are: radio receiver, keyboard, microphone, photo/video camera, accelerometer. Examples of device output interfaces are: radio sender, speaker, display, LED, vibrating motor.

Figure (1.1) shows the device setup. I name the two involved devices *Alice* (A) and *Bob* (B). The legitimate user of the devices A and B (e.g. their owner) is simply called *user*. The term *entity* is used to refer to a device or a human and the term *adversary* or *attacker* is used for a malicious entity whose aim is to prevent the user from achieving his goal. To avoid using "he/she" I assume, without loss of generality, that the adversary is male and the user is female.

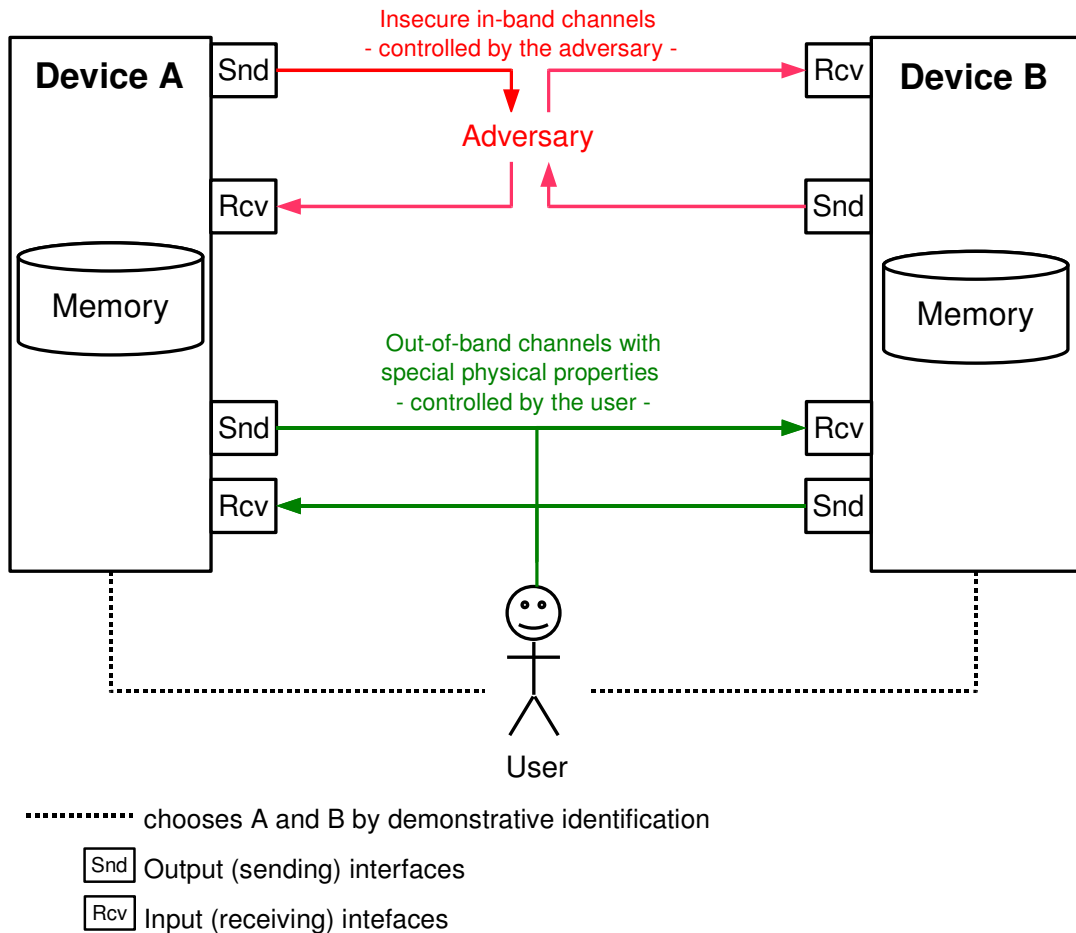


Figure 1.1: Device Setup

The user can choose the devices she wants to pair (black dotted line in Figure 1.1). In most of the device pairing protocols she does this by a *demonstrative identification* [1] of the devices. Demonstrative identification is the method of selecting the devices, a user wants to pair, based on a physical context (e.g. the printer in front of me, all handheld's on this table).

1.4.1 Channels

A channel is a capability for communication. It can be used to exchange messages between two entities. In our device setup a channel connects an input with an output interface of two entities. Note that the user is also an entity and can send or receive data. I use the term *in-band channel* for a fast but insecure and unreliable channel and the term *out-of-band (OOB) channel* for a usually low bandwidth channel that guarantees some kind of security (e.g. confidentiality, message integrity) to the communicating entities. Typically *out-of-*

band implies that there is an *in*-band channel available. An in-band channel can for example be a connection between two computers in an insecure WLAN network and an out-of-band channel could be a user who exchanges some secret information between the two computers using a memory stick.

I use the following terms to refer to security properties of channels used in device pairing:

Insecure Channel from A to B is a channel that provides no security. An attacker can read, insert, modify or delete messages on this channel.

Authentic Channel from A to B is a channel that provides authenticity, meaning that only *A* can send on this channel. An attacker can read but not insert or modify messages on this channel.

Secure Channel from A to B is a channel that provides authenticity and confidentiality, meaning that only *A* can send and only *B* can read from this this channel. An attacker can neither read, insert nor modify messages on this channel¹

1.4.1.1 Device Memory

I assume that all data of a device (e.g. keys) is stored in its memory and that only the device itself can directly access its memory. Other entities can only send requests to an input interface and wait for a response from an output interface. A device can execute functions over the memory data. The supported set of functions is limited by the hardware (e.g. microprocessor). I further assume that the devices can generate random data (e.g. by receiving random data from the environment).

1.5 Security Relation

A *security relation* is a special association between two devices, due to their memory content. We consider the following security relations for a device *A* and *B*:

$A \text{---} B$ Secret key shared by *A* and *B*. *A* and *B* have a key that is not known to anybody else. An adversary is not able to predict this key.

$A \text{---} B$ *A* has an authentic copy of *B*'s public key.

$A \text{---} B$ *B* has an authentic copy of *A*'s public key.

$A \text{---} B$ *A* has an authentic copy of *B*'s public key and *B* has an authentic copy of *A*'s public key.

Note that the symbol --- is also used for authenticated Diffie-Hellman half-keys (a.k.a. Diffie-Hellman public keys)

¹This implies that an adversary can also not delay or replay a message.

It is possible that a user knows a secret key after the device pairing process, for example because she transmits the key from A to B . As I consider the user as a separate entity (not part of A or B) the key would then become a shared secret between A , B and the user. Nevertheless I use the notion $A \bullet \text{---} \bullet B$ in this case ².

1.5.1 Creation/Deletion of Security Relations

A security relation can be established using a device pairing protocol when both devices fulfill necessary conditions (e.g. have the necessary interfaces) and the user is in an environment where he can execute the protocol.

1.6 Definition of Device Pairing

Device pairing, also known as *secure device pairing*, *device association*, *secure first connect*, *secure initialization* or simply *pairing*³ denotes the creation of a security relation between devices A and B , described in the device setup (1.1). I assume that A and B do not have any prior context in common nor do they share any common point of trust. We can therefore not profit from pre-shared secrets, authenticated public keys, certification authorities or a public key infrastructure.

I say that A and B are *paired* if there exists at least one security relation between them and *unpaired* if no security relation exists between them. I assume that a security relation exists until the user explicitly deletes it.

Note that I restrict myself to the case of pairing two devices, but that one could think of protocols where more than two devices are associated in a single protocol run. An example of such a protocol would be, when you put your Bluetooth mobile phone and 3 different Bluetooth headsets, which you want to use with your phone, in a Faraday cage. The mobile phone generates a secret key and broadcasts it using a radio signal to the headsets. In this case the term *pairing* would be inappropriate since the word implies that exactly two entities are involved.

1.7 Trade-Off Between Security and Usability

Device pairing protocols often have to make a trade-off between security and usability. In string comparison protocols for example, the user has to compare two strings displayed by A and B . The strings are typically hashes of a previously transmitted public key. If they are equal the user accepts the protocol run, otherwise she has to abort it. In this case it is

²We could imagine of that the user does not memorize or immediately forgets the key after device pairing.

³In this paper, I use the term *device pairing* to refer to the problem at hand.

possible that an adversary tries to fool the user. If the strings are too short, an adversary could probably successfully attack the protocol. If they are too long the protocol is no longer user-friendly. There are a lot of other device pairing protocols where we have to make a trade-off between security and usability. Since this is not a usability study, I will not consider the usability of device pairing protocols here.

Another important point, not addressed in this work, is that a protocol should be transparent for the user. This means that the user should be intuitively aware why she does something in the protocol and what she achieves with that. This topic is also not addressed in this work.

When you know that you're capable of dealing with whatever comes, you have the only security the world has to offer.

(Harry Browne)

2 Use Cases

In this chapter we consider a few application scenarios where device pairing is needed. The following list shows a set of devices that somebody could like to pair.

- PC/Laptop
- Computer mouse/keyboard
- Handheld/PDA (Personal Digital Assistant)
- Cellphone
- Media player
- Headset
- GPS receiver
- Wireless hard disk
- Camera
- Printer/Copier/Fax machine
- Smart watch
- Hi-Fi device
- Radio remote control
- Access point
- Household appliance
- Radio equipment/Walkie-talkie

Table 2.1: Potential devices for pairing

These devices could be equipped with a wireless adapter to communicate with each other. Examples of wireless data standards for personal/local area communication are:

- Wi-Fi, (IEEE 802.11)
- Bluetooth
- Wireless USB (WUSB)
- Wibree
- ZigBee
- Ultra-wideband (UWB)

Table 2.2: Personal/local area wireless data standards

Now we consider concrete application scenarios where we need to generate a security relation between two unpaired devices. A sufficient security relation is stated on the left side.

Headset ●——● **Mobile phone** You want to connect a Bluetooth headset to your mobile phone. An adversary should not be able to overhear your telephone conversation. Imagine that the adversary can not directly listen to your voice (e.g. because he is not in the same room), but he is within the Bluetooth signal range.

PC ●——● **Wireless hard disk** You want to connect your wireless hard disk to your PC. An adversary should not be able to overhear or insert/modify transmitted data.

Keyboard,mouse ●——● **PC** You bought a new wireless keyboard and mouse for your PC. An adversary should not be able to overhear or insert/modify the transmitted characters (e.g. passwords).

Handheld ●——● **Laptop** You meet a friend at a public place and want to exchange some secret documents between your handheld and his laptop. An adversary should not be able to read the transmitted documents.

Laptop ●——● **Access Point** You want to enroll your new laptop into your home WLAN. More precisely, you want to generate a shared secret key between your laptop and the access point to encrypt all further communication.

Laptop ●——● **Printer** You want to print a secret document with your laptop on a specific WLAN printer in a public area.

Your handheld ●——● **Your friends handheld** You meet a friend at a public place and want to receive his digital visiting card together with his PGP¹ public key on your handheld. An adversary should not be able to fake your friends PGP public key during transmission.

Radio remote control ●——● **Home cinema device** You want to control your home cinema devices (TV, beamer, DVD player, etc.) with an universal radio remote control. An adversary (e.g. your neighbor) should not be able to (un)-intentionally control your cinema devices.

GPS receiver ●——● **Camera** You carry a photo camera and a GPS receiver with you and want to tag each picture you take with the actual GPS coordinates received from the GPS device. An adversary should not be able to fake the GPS coordinates.

Note that it can make sense to generate a shared secret key between two devices, even if an authentic copy of a public key would be sufficient. This is because symmetric cryptography is less power consuming and is much faster than asymmetric cryptography.

We can also think of scenarios where a provider wants to get some special guarantees from the customers of his service or vice versa:

Assume that the provider of a public access point in a restaurant wants to restrict the use

¹PGP: Pretty Good Privacy. A public key encryption program originally written by Phil Zimmermann in 1991.

of his access point, such that only customers inside the restaurant can use the Internet (and therefore need to consume something). Restricting the transmission power of the access point does not fulfill this requirement, because the access point provider does not know how large the antennas of the customer devices are and therefore how low the transmission power of the access point should be.

Assume a bank publishes the current stock prices in a window display, but also broadcasts this information with a short range radio transmitter, such that visitors can directly receive the stock prices on their handheld. A customer of this service can only trust these prices when the underlining protocol ensures that the data has not been modified during transmission and that the sender is indeed the bank.

No serious commentary will say that the user has no responsibility. We all have responsibilities to lock our doors in our homes and to buckle up when we get in cars.

(unknown)

3 Device Pairing Protocols

In this chapter I describe a set of different device pairing protocols. The classification of the protocols is shown in Figure (3.1). This classification is a modified version of Suomalainen, Valkonen and Asokan's [23] classification of key agreement protocols, shown in Figure (3.2). Since we do not only consider shared secret keys as security relations, we extend our classification by the security relation $\bullet \text{---} \bullet$ (authentic copy of a public key). Beside that, we introduce the protocol classes P2-P4 and skip all insecure classes (unauthenticated key agreement) those that need a pre-shared secret key (authenticated key agreement based on symmetric cryptography) because in our device setup a pre-shared secret key is not available.

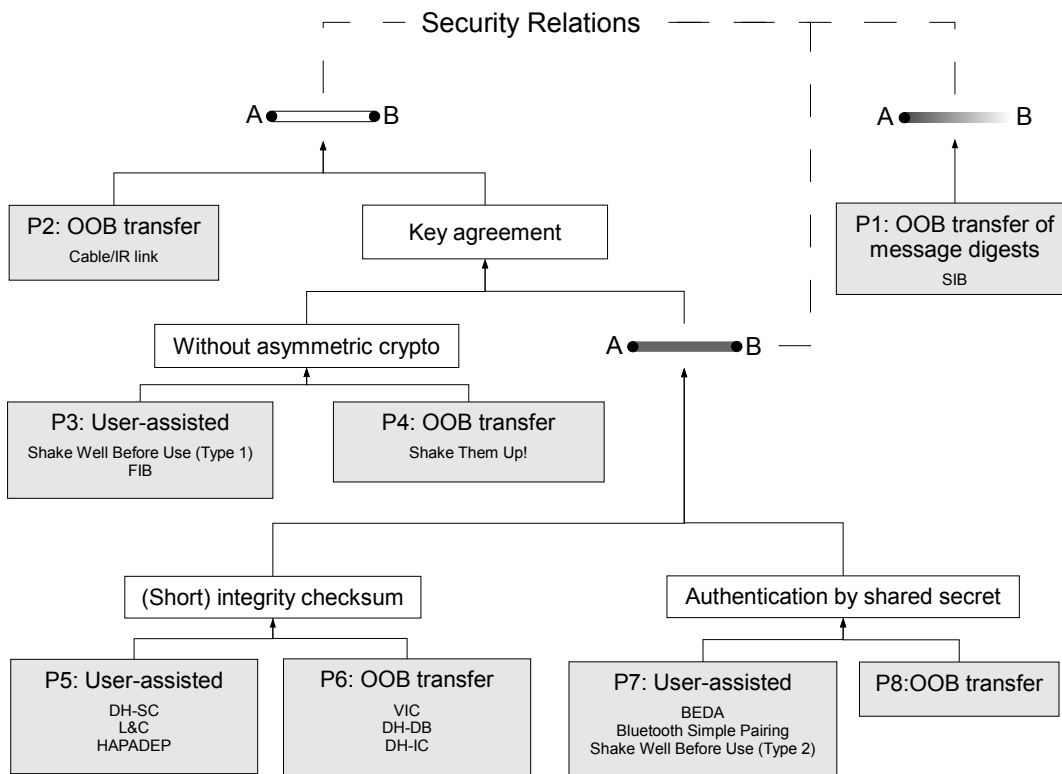


Figure 3.1: Classification of device pairing protocols

I classify the selection of device pairing protocols into 8 protocol classes (P1-P8), shown in Figure (3.1). The protocol name or a shortcut of the protocol name is stated in the gray rectangles. Each protocol class and superclass (white rectangles) is described in a separate section in this chapter and the concrete protocols are described in subsections of these.

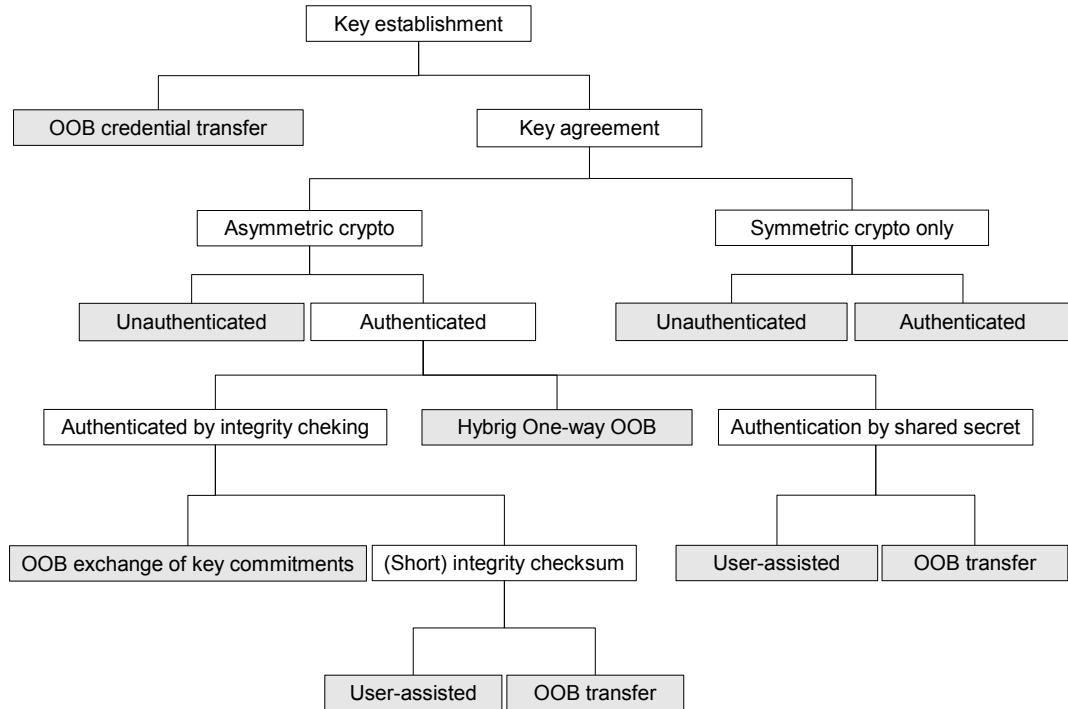


Figure 3.2: Classification of Key Agreement Protocols [23]

3.1 Notations

We use the following notations in the protocol descriptions:

PK_X	Public key of device X (also used for Diffie-Hellman half-keys).
PK_X^{-1}	Private key of device X .
K_X	Symmetric key generated by device X .
\oplus	Bit-wise xor operator.
$ $	Concatenation operator.

A “hat” above a value, e.g. \hat{PK}_A , denotes that the value could be modified (by an adversary) during transmission.

3.2 General Protocol Assumptions

A device is compromised when unauthorized individuals have gained access to it. We assume that A and B are not compromised before executing the pairing protocol, otherwise the involved entities get no reliable guarantees from the protocol. We further assume that A and B trust each other, otherwise it makes no sense to create a security relation between them.

For all hash functions, denoted as $h()$, we assume second pre-image resistance, i.e. the probability of finding another input string that hashes into the same value is negligible. We apply $h()$ exclusively to public keys in this work and assume that $h()$ is at least 48 bits for ephemeral public keys and 80 bits for long-term public keys [18]. Further note that an adversary has usually no benefit of knowing a second preimage of a hashed public key, if he does not know the corresponding private key as well.

3.3 General Attacker Model

For all in-band channels we assume a standard Dolev-Yao attacker model [8], in which the adversary controls the communication. He can therefore eavesdrop, block, delay, modify or inject messages on in-band channels. For all radio channels (in-band and out-of-band) we assume that the adversary can jam the transmission and therefore prevent transmission of information. We further assume that the user has exclusive access to the keyboard and buttons of A and B during device pairing.

3.4 One-Way Public Key Authentication

A one-way public key authentication protocol can be used to create a security relation $A \dashv\dashv B$ or $A \dashv\dashv B$ if A and B supports the protocol requirements. A security relation like $A \dashv\dashv B$ can be used by A to encrypt messages for B or to verify the authenticity of messages signed by B .

Since a security relation $\dashv\dashv$ includes the security relation $\dashv\dashv$, we consider here only protocols that are unable to create a security relation $\dashv\dashv$ in a single protocol run. Protocols which create $\dashv\dashv$ are described in Section (3.12).

There are many use cases where the creation of such a “unilateral” association between two devices is sufficient (e.g. our printer example in 1.3). Nevertheless, most of the device pairing protocols are designed to create a “bilateral” association like a shared secret key, because then, both devices can use fast and CPU non-intensive symmetric cryptography to secure their communication. Besides that, the creation of a shared secret key is usually only slightly more difficult than creating a security relation like $A \dashv\dashv B$ if the protocol can roll out some tasks to the user and the user has full access to both devices, which is usually the case for device pairing. We will see this by comparing the following protocol for one-way public

key authentication with an extended version, described in Section (3.15.2) that can be used to create a shared secret key.

Note that all protocol classes for both-way public key authentication (P5-P8) plus the class P2 can also be used for one-way public key authentication. We do not consider these protocols here because we can achieve a higher goal with these protocols. We only consider the protocol class P1, *OOB transfer of message digest*, for one way public key authentication.

3.5 OOB Transfer of Message Digests (P1)

Protocols in class P1 use an authentic OOB channel to transfer a public key or a hash of a public key (also called *commitment to a public key* [1]) from device A to B . The authenticity of the channel is usually guaranteed by a combination of physical channel properties, the protocol and the user. If the OOB channel is suitable to transfer a public key, device A can simply send the public key over the OOB channel to B . If we can only send small messages (e.g. 60 bit) over the OOB channel we can use the channel to authenticate a message sent over an insecure (high bandwidth) channel. To do this, A sends her public key PK_A over the insecure channel and a hash $h_A = h(PK_A)$ of that public key over the (low bandwidth) authentic OOB channel to B . After B receives both messages, he can check the authenticity of the received public key $\hat{P}K_A$ by comparing $h_A \stackrel{?}{=} h(\hat{P}K_A)$. If they match, B accepts A 's public key, otherwise he rejects it.

3.5.1 Channel Properties

The attacker can read but not write on the OOB channel.

3.5.2 Seeing-is-Believing

Seeing-is-Believing (SiB) proposed by McCune *et al.* [15] uses camera phones for one-way public key authentication. Today's camera phones provide sufficient capabilities to scan 2D barcodes. Having such a camera phone and a device with a good enough screen to display 2D barcodes yields a visual OOB channel. This channel guarantees to the user that the barcode can only be sent by the device the user took a picture from.

3.5.2.1 Attacker Model

The attacker can read but not write on the visual channel (passive attacker). The attacker is computationally bounded.

3.5.2.2 Protocol

Figure (3.3) shows the SiB protocol where B authenticates A 's public key PK_A . In the first step Alice displays the hash $h(PK_A)$ of her public key encoded as a 2D barcode on her display. $h(PK_A)$ is then transmitted to B using the visual OOB channel. To do that, Bob takes a picture of Alice's display and decodes the 2D barcode in the picture. In the second step Alice sends the public key PK_A over the insecure in-band channel to B where B can check it's integrity using the previously received hash (h_A). To avoid problems of user-tracking new keys can be generated for each pair of devices.

- 1 $A \rightarrow B$ (visual channel): $h_A = h(PK_A)$
- 2 $A \rightarrow B$ (insecure channel): PK_A
 B verifies $h_A \stackrel{?}{=} h(\hat{PK}_A)$. If they match, B accepts \hat{PK}_A .

Figure 3.3: SiB one-way authentication protocol. B authenticates A 's public key.

3.5.2.3 Device Configurations

SiB supports several device configurations. When only one device has a camera and the other device has only a display, then only the camera device can authenticate a message from the display device and not vice versa. When both devices have a camera and a display, both devices can authenticate a message from the other device by performing the protocol twice in both directions.

When only one device has a camera and the other device has neither a camera nor a display we are not lost. We can then print a long-term public key hash encoded as a 2D barcode on a sticker and attach the sticker to the housing of the display-less device. To authenticate the public key of the display-less device, the user simply takes a picture of the sticker and performs the SiB protocol as usual.

3.5.2.4 Assumption

We assume that an adversary can not fool the user into taking a picture with B 's camera of a barcode that does not belong to A . For stickers, this means that an adversary can not add (e.g. paste over) or modify the sticker without being noticed by the user or the camera device.

3.5.2.5 Device Requirements

For using ephemeral public keys, B has to be equipped with a good enough camera to display 2D barcodes that encode at least 48 bits of data.

3.5.2.6 What the Protocol Achieves

At the end of a successful protocol run B guarantees that the public key PK_A it received was sent by the device that has been photographed.

3.5.2.7 Limitations and Drawbacks

Missing or low-quality cameras and display or an environment with little light can limit the use of SiB. SiB can further not be used when the devices, to be paired, can not be sufficient near to each other. Stickers prevent using freshly generated public keys and increase the risk that an adversary fools a user into accepting a wrong public key by replacing or modifying the sticker.

3.5.2.8 Channel Bandwidth

A typical barcode has dimensions approximately $2.5 \times 2.5\text{cm}^2$ to allow recognition from a reasonable distance and consists of a total of 83-bits of information (15-bits are for *forward error correction*) [18]. This allows us to transfer at most 68 bits with a picture.

3.5.2.9 Presence Property

We assume for SiB that only the device in front of B 's lens can send a message. A on the other hand, gets the guarantee that B must be *present* at time t to read a message display on A at time t . This channel property is called *presence*. The following protocol example uses presence to pair two devices that have no camera.

Assume you have two devices, a DVD player and a TV, that communicate wirelessly. The TV needs an authentic copy of the DVD's public key, to verify the DVD's signature on the data stream. Since the TV has no camera, we can not directly establish a visual channel from the DVD player to the TV. In this case, a camera device P (e.g. a camera phone), can be used to forward a hash of the DVD's public key to the TV. Consider the following protocol where HMAC is a *keyed-Hash Message Authentication Code*.

The first two steps of the protocol are the same as in the basic SiB protocol. At the end of step two, the camera device has an authentic copy of the DVD's public key. In step 3, the TV sends a secret key K_{TV} encoded in a 2D barcode over a visual channel to the camera phone. Here we use the presence property of the channel. The user has to take care, that nobody except P can read the secret key on the TV 's display. This can be seen as a realization of a secure channel. The camera phone can now use the secret key to sign the DVD's public key ($HMAC_{K_{TV}}(PK_{DVD})$) and send both, the signature and the public key, to the TV. In the last step, the TV verifies the signature of the public key and accepts the public key if and only if it is correct.

- 1 $DVD \rightarrow P$ (visual channel): $h_{DVD} = h(PK_{DVD})$
- 2 $DVD \rightarrow P$ (insecure channel): PK_{DVD}
 P verifies $h_{DVD} \stackrel{?}{=} h(\hat{P}K_{DVD})$. If they match, P accepts $\hat{P}K_{DVD}$.
- 3 TV generates a secret key K_{TV}
 $TV \rightarrow P$ (visual channel): K_{TV}
- 4 P calculate $t = HMAC_{K_{TV}}(PK_{DVD})$
- 5 $P \rightarrow TV$ (insecure channel): PK_{DVD}, t
 TV verifies $t \stackrel{?}{=} HMAC_{K_{TV}}(\hat{P}K_{DVD})$. If they match, TV accepts $\hat{P}K_{DVD}$.

Figure 3.4: Presence protocol example. At the end of the protocol, the TV has an authentic copy of the DVD's public key.

3.6 Shared Secret Key Generation (P2-P8)

Protocols in class P2-P8 can be used to create a security relation $A \bullet \text{---} \bullet B$. We differ between *OOB key transfer protocols* and *key agreement protocols*.

3.7 OOB Key Transfer Protocols (P2)

In *OOB key transfer* protocols, one device chooses a random key and sends it over a (physically) channel to the other device. Examples of such channels are: a cable, an infrared link or a user who transfers data between two devices using a memory stick. The attacker can be computationally unbounded in this case.

3.7.1 OOB Channel Properties

The attacker can neither read from nor write to the OOB channel.

3.8 Key Agreement Protocols (P3-P8)

In contrast to key transfer protocols (P2), in key agreement protocols both devices A and B participate in the generation of a shared secret key. We distinguish between asymmetric key agreement protocols (e.g. Diffie-Hellman key agreement) and key agreement protocols that do not need public key techniques (e.g. pure symmetric key agreement schemes).

3.9 User-Assisted Key Agreement (P3)

Protocols in class P3 need the user to exchange secret information between A and B or to derive random data that is used for key agreement.

3.9.1 Channel Assumptions

The attacker can neither read from nor write to the user-assisted channels.

3.9.2 Shake Well Before Use (type 1)

Shake Well Before Use proposed by Mayrhofer and Gellersen [14] uses accelerometer data to establish a shared secret key between two devices equipped with accelerometers. The accelerometer data is generated by shaking the devices together in one hand. The paper describes two different protocols, on how to generate a shared secret key from the accelerometer data. We consider only the first protocol here (we call it type 1). The second protocol (type 2) is described in the protocol class *authentication by shared secret key (P7,8)* (Section 3.16).

3.9.2.1 Attacker Model

The attacker can not predict or derive acceleration data that the shaken devices receive. The attacker is computationally bounded.

3.9.2.2 Device Requirements

All devices must be equipped with accelerators and sufficient accurate real-time clocks for temporal data alignment.

3.9.2.3 Protocol

In protocol type 1 of Shake Well Before Use [14], both devices directly derive a key from the acceleration patterns. This protocol is very dynamic since the user simply needs to shake the devices as long as random input is needed. To derive a key from the accelerator data both devices first extract feature vectors from the raw data. Since these vectors usually differ on both devices, an additional protocol to synchronize both devices to the same key is needed. This approach can also be used to generate a shared secret key between more than two devices.

3.9.2.4 Assumptions

We assume that the user shakes the devices as she likes. This ensures that the generated acceleration data is fairly random. We further assume that the protocol to synchronize the devices to the same key, does not reveal any useful information for the adversary to construct the whole or parts of the secret key.

3.9.2.5 What the Protocol Achieves

At the end of a successful protocol run, the devices that are shaken together, established a shared secret key.

3.9.2.6 Overall Completion Time

According to [14], 20 seconds of shaking should be sufficient to generate a shared secret key of 128 bits.

3.9.3 Feeling-is-Believing

Feeling-is-Believing (FiB) proposed by Buhan *et al.* [3] provides an OOB channel which uses biometrics to generate a shared secret key.

3.9.3.1 Attacker Model

The attacker can not fool a biometric reader into accepting copies of biometrics. The attacker is computationally bounded.

3.9.3.2 Device Requirements

Both devices are equipped with a biometric reader (e.g. fingerprint reader).

3.9.3.3 Protocol

The protocol is shown in the following Figure:

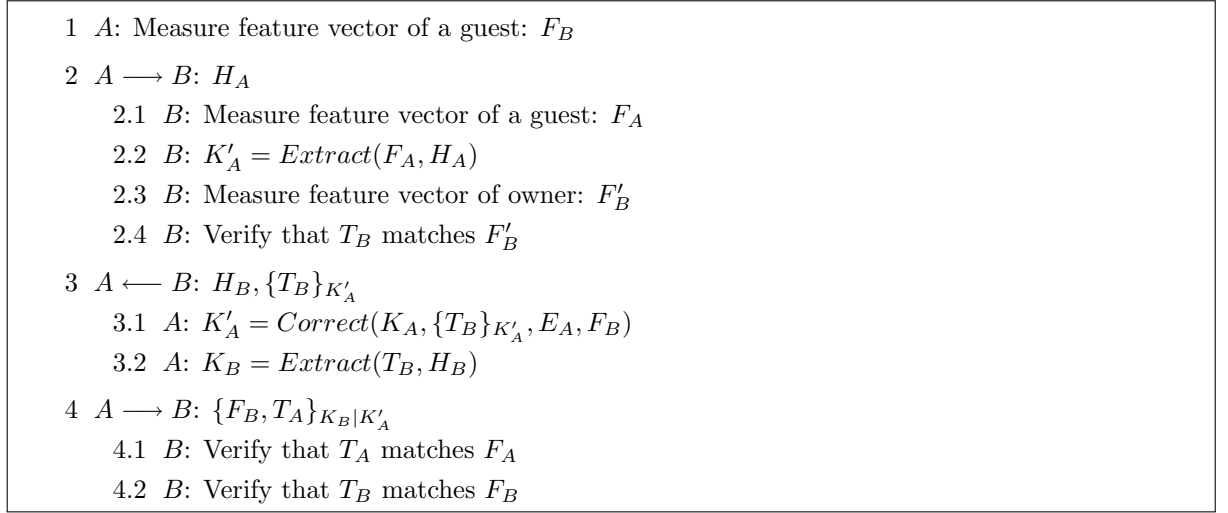


Figure 3.5: FiB Protocol [3].

We use the following notation:

D_x	Identifies device x
$T_x \in \mathbb{R}^n$	The enrolled template of user x
$H_x \in \mathbb{R}^n$	Helper data of user x
$F_x \in \mathbb{R}^n$	Feature vector currently measured for user x
$K_x \in \{0, 1\}^n$	Key extracted from T_x and H_x
$K'_x \in \{0, 1\}^n$	Key extracted from F_x and H_x
$E_x \in \langle \mathbb{N}, \mathbb{R} \rangle^n$	Error profile for a user x

The both-way OOB channel between A and B is realized by measuring the feature vector (F_x) of user B on device A (step 1) and vice versa (step 2.1). This can be seen as transmitting the feature vector from one device to the other. The second half (K'_A) of the secret key is derived from the feature vector of user A (step 2.2) and the first half (K_B) of the secret key is derived from the template of user B (step 3.2). Unfortunately, the measurement of biometrics is imprecise. We therefore would not get the same key on both devices if we simply extract the key from the measured feature vector F_X or the user template T_X , respectively. The protocol uses a cryptanalysis function (*correct()*) to solve this problem.

After exchanging both feature vectors (step 1 and 2.1) device B extracts a uniformly bit string K'_A from the feature vector F_B and the helper data H_A using a fuzzy extractor function (*Extract()*). K'_A is used in step 3 to encrypt the template T_B which is then sent to A . A can not directly decrypt $\{T_B\}_{K'_A}$ because the key K_A and K'_A might differ due to biometric reading inaccuracies. Because K_A and K'_A only differ in a few bits, we can

use the cryptanalysis function *correct()* to derive the correct key K'_A . Note that only the communicating devices are able to perform this analysis, because they can start on key material that is almost correct. An adversary has to start from random key material, and should thus take longer to derive the correct key K'_A . The key is then used to decrypt B 's template T_B from $\{T_B\}_{K'_A}$. In step 3.2, A extracts the second part K_B from B 's template T_B . The shared secret key between A and B is then $K_B|K'_A$.

3.9.3.4 Assumptions

We assume that devices A and B hold a biometric data measurement (T_A, T_B) from their owners before the protocol starts.

3.9.3.5 What the Protocol Achieves

At the end of a successful protocol run, A (B) guarantees that the generated secret key is only known to B (A) and that the owners of device A and B provided their necessary biometrics. The users A and B must therefore be present to generate a shared secret key between device A and B .

3.10 OOB Key Exchange (P4)

Devices in protocol class P4 provide a both-way OOB channel. This channel is used to exchange secret information, the devices use for key agreement.

3.10.1 Channel Properties

The attacker can not read on the OOB channel. If he can write on the OOB channel depends on the concrete protocol.

3.10.2 Shake Them Up!

Shake Them Up! proposed by C. Castelluccia and P. Mutaf [6] uses anonymous broadcasts to establish a shared secret key between A and B .

3.10.2.1 Attacker Model

We assume the attacker controls the communication with the exception that he can not determine the origin of a message. If he inserts a packet with source field set to A , it will be interpreted as 0 on A and as 1 on B . If he inserts a packet with source field set to B ,

it will be interpreted as 1 on A and as 0 on B . The attacker can further drop packets, but he does not know who was the sender of such a packet. The adversary can therefore not explicitly prevent only A or only B from sending packets. The attacker is computationally (For the key agreement phase it can be computationally unbounded).

3.10.2.2 Protocol

Figure (3.6) shows a sample protocol run in which an 8 bit key is established between A and B .

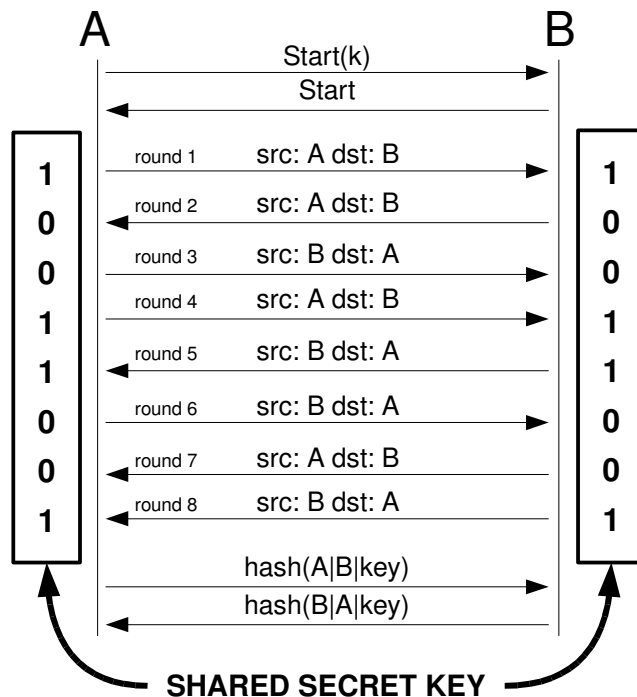


Figure 3.6: Key agreement protocol for movement-based pairing [6].

The protocol is randomly initiated by one of the devices with a start message containing the desired key length (k). The other device answers with an empty (payload) start message. Both start messages contain the correct sender address in the source field. A shared secret key can now be generated bit by bit. In every protocol round either A or B sends a single bit to the other device. A sends the bit 1 (0) to B by broadcasting an empty packet with the source field set to A (B). B sends the bit 1 (0) to A by broadcasting an empty packet with the source field set to B (A). Note that A and B stands here for A and B 's address. When a device receives a foreign packet (a packet that is not sent by itself) it decodes it to 1 if the source field is set to the others device address and 0 if the source field is set to its own device address. At the end of the protocol, both devices exchange a hash of the claimed shared secret key to check whether both devices agreed on the same key. Here we

require that the attacker is computationally bounded. These messages are also exchanged in random order.

The protocol security mainly relies on the following three properties. The first two properties guarantee the anonymous broadcast channel and the third property prevents MitM attacks.

Spatial indistinguishability ensures the anonymity of the broadcast channel by preventing spatial signal analysis (e.g. signal power analysis). This property can be guaranteed by moving the devices around each other during protocol execution.

Time indistinguishability ensures the anonymity of the broadcast channel by preventing timing analysis. This property is ensured, if in each round of the protocol, the probability that A or B sends a packet is $1/2$. This can be realized by sending the packets at a randomly chosen point in time within a fixed time interval. A media access protocol like CSMA¹ would support this.

Start message authentication is needed to prevent a MitM attack. This attack is explained in Section (4.3.2). To authenticate the start messages, the user could be asked to validate the device addresses. Since this is not very user-friendly and would need an appropriate output interface (e.g. display) on both devices, Shake Them Up! proposed another protocol that uses proximity for authentication. I refer to [6] for this protocol.

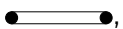

3.10.2.3 Assumptions

We assume that both devices have an authentic copy of the other's device address and that the user moves A and B during pairing around each other to ensure spatial indistinguishability. Further, A or B discard the generated n bit key if they have not both contributed at least e.g. $n/3$ bits to the key. Finally we assume that $hash()$ is a preimage resistant hash function. I will explain the last two assumptions, in more detail in Chapter 5.

3.10.2.4 What the Protocol Achieves

At the end of a successful protocol run, A and B established a shared secret key.

3.11 Diffie-Hellman Key Agreement (P5-P8)

Diffie-Hellman (DH) key agreement [7] can be used to create a security relation , on the basis of a security relation .

¹CSMA: Carrier Sense Multiple Access (e.g. used in WLAN IEEE 802.11).

3.11.1 Protocol

The protocol is shown in Figure (3.7). This protocol is based on modular exponentiation and on the difficulty of the discrete logarithm problem. Since DH key agreement is vulnerable to man-in-the-middle attacks we need to authenticate the exchanged half-keys. This protocol allows us therefore to reduce the problem of creating a shared secret key between A and B to the problem of creating a security relation $A \bullet \bullet B$. Device pairing protocols in the classes P5-P8 can be used to create this security relation.

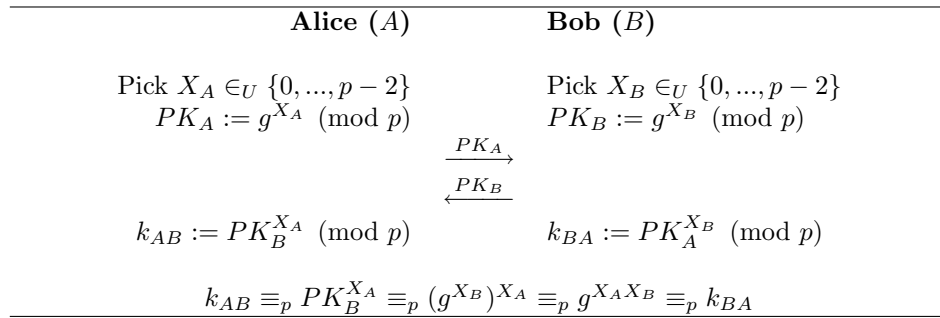


Figure 3.7: The Diffie-Hellman key agreement protocol. The prime p and the generator g are publicly known parameters.

3.11.2 Assumptions

In order for the shared key $k_{AB} = k_{BA}$ to be secure, the so called *Decisional Diffie Hellman (DDH)* assumption must hold. The DDH-assumption holds in the group G if, for uniformly and independently chosen $a, b, c \in \{0, \dots, |G| - 1\}$, the triples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable.

3.12 Both-Way Public Key Authentication

In the context of device pairing, both-way public key authentication is the problem of creating the security relation $A \bullet \bullet B$. This can be done by executing a one-way public key authentication protocol twice. First, A authenticates B 's public key, afterwards they change the roles and B authenticates A 's public key (or vice versa). Protocols in class P1 (OOB transfer of message digest) can therefore be used for both-way public key authentication by establishing two unspoofable OOB channels in both directions. As described for Seeing-is-Believing this can not be done if one device has not the necessary interface to establish the OOB channel. All protocols in this section can achieve both-way public key authentication even if only a one-way OOB channel is available (not that we use the term OOB channel only for device-to-device channels). All these protocols first exchange the public keys over

insecure in-band channels and then authenticate these public keys by use of an OOB channel and/or by user-assistance.

3.13 Authentication by Integrity Checksum (P5,P6)

Protocols in class P5 and P6 create, after exchanging public keys, a short integrity checksum on A and B . These checksums are compared either by the user (P5) or by one or both devices (P6). The protocol guarantees, if the checksums are equal, that the public keys were exchanged by the devices which sent the integrity checksums to the comparison oracle (e.g. user). In all protocols here, the user (intuitively) identifies A and B as the sending devices of the integrity checksums. This guarantees that indeed A (B) is the sender of the public key received by B (A).

3.14 Authentication by Integrity Checksum - User-Assisted (P5)

Protocols in this class need the user to verify if two integrity checksums are equal or not. If they are equal, the user accepts the public key exchange by pushing an accept button on both devices. Otherwise she rejects the public keys by pushing a reject button on both devices.

3.14.1 Assumption

We assume that the user correctly compares the two integrity checksums, displayed by A and B and that she pushes the right feedback button on A and B .

3.14.2 Device Requirements

Both devices have an input interface for the user's feedback (e.g. keyboard) and an output interface to display the integrity checksum (e.g. display, speaker).

3.14.3 Diffie Hellman Key Agreement with String Comparison

Diffie Hellman key agreement with string comparison (DH-SC) proposed by Čagalj and Čapkun [24] provides an optimal trade-off between the length of the integrity checksums and the computational effort an attacker has, to break the authentication scheme.

3.14.4 Attacker Model

The attacker can read from all involved channels but he can not write on the visual channel from the devices to the user. The attacker is further computationally bounded.

3.14.4.1 Protocol

The protocol is shown in Figure (3.8). The security parameter k determines the length of the integrity checksum and can be chosen by the user. The goal of the protocol is to authenticate the Diffie-Hellman half-keys PK_A and PK_B . This protocol is used as a building block for other pairing protocols (3.15.3, 3.15.4). In the simplest application, which we consider here, both devices show the integrity checksums i_A and i_B on their display. The user compares the two checksums and pushes an accept/reject button on both devices depending on whether the checksums are equal or not.

Encoding

Besides the length of the integrity checksums, the encoding plays an important role on how user-friendly the protocol is. To encode a for example $k = 60$ bit checksum by an alphabet of $n = 26$ symbols, the user would need to compare at least $\lceil k/\log_2(n) \rceil = 13$ symbols. To make the comparison easier we could encode the checksum to l short words. If we choose $l = 4$, then we need to store $2^{k/l} = 4096$ different words in a directory on both devices. Other ideas are to encode the checksum to a syntactically-correct sentence (3.14.5) or to a sequence of pictures/pictograms.

Protocol	
<p style="text-align: center;">Alice</p> <p style="text-align: center;">Given ID_A, PK_A</p> <p style="text-align: center;">Pick $N_A \in_U \{0, 1\}^k$</p> <p style="text-align: center;">$m_A \leftarrow 0 ID_A PK_A N_A$</p> <p style="text-align: center;">$(c_A, d_A) \leftarrow commit(m_A)$</p>	<p style="text-align: center;">Bob</p> <p style="text-align: center;">Given ID_B, PK_B</p> <p style="text-align: center;">Pick $N_B \in_U \{0, 1\}^k$</p> <p style="text-align: center;">$m_B \leftarrow 1 ID_B PK_B N_B$</p> <p style="text-align: center;">$(c_B, d_B) \leftarrow commit(m_B)$</p>
	$\xrightarrow{c_A}$
	$\xleftarrow{c_B}$
	$\xrightarrow{d_A}$
$\hat{m}_B \leftarrow open(\hat{c}_B, \hat{d}_B)$	$\xleftarrow{d_B}$
Verify 1 in \hat{m}_B ; $i_A \leftarrow N_A \oplus \hat{N}_B$	$\hat{m}_A \leftarrow open(\hat{c}_A, \hat{d}_A)$
	Verify 0 in \hat{m}_A ; $i_B \leftarrow N_B \oplus \hat{N}_A$
If $i_A = i_B$ Alice and Bob accept \hat{m}_B and \hat{m}_A , respectively.	

Figure 3.8: DH-SC: Mutual Message Authentication using Short String Comparison [4].

3.14.4.2 Assumptions

We assume that the used commitment scheme $\langle commit(), open() \rangle$ is ideal (hiding and binding)².

3.14.5 Loud-and-Clear

Loud-and-Clear (L&C) proposed by Goodrich *et al.* [10] describes four protocol variants for one-way and both-way public key authentication. In all variants the user compares two syntactically-correct (english-like) sentences, generated from hashes of previous exchanged public keys. The protocol variants differ in the way the user compares the two sentences. In type 1, both devices vocalize the sentence using an audio speaker. In type 2 and 3, one device vocalizes the sentence and the other device displays it. This approach can be used if only one device has an audio speaker. In type 4, both devices display the sentence. Last type is useful when both devices lack an audio speaker. In all four variants, the user compares the two vocalized/displayed sentences and accepts the protocol run if they are equal.

3.14.5.1 Device Requirements

Protocol type 1: Both device are equipped with audio speakers.

Protocol type 2: A is equipped with an audio speaker and B with a display.

Protocol type 3: B is equipped with an audio speaker and A with a display.

Protocol type 4: Both device are equipped with displays.

3.14.5.2 Protocol

Figure (3.9) shows protocol type 3 of L&C for on-way public key authentication. Both-way public key authentication can for example be achieved by applying DH-SC (3.14.3) and generating sentences from the integrity checksums i_A, i_B . To ensure the authenticity of the integrity checksums, the user further needs to verify that the sentences are indeed originated from A and B (she has to guarantee that the right devices are playing).

3.14.5.3 Protocol Assumptions

We assume the adversary can not overlay the audio signal to fool the user into comparing two integrity checksums that are not sent by A and B . We further assume that for a given sentence $s = f(h(PK))$, where $h(PK)$ is the hash of a public key and $f()$ is the function that maps a hash to a sentence, the adversary can not find a value PK' with $PK' \neq PK$ and $f(h(PK')) = s$.

²Note that in practice, a commitment scheme can not be perfectly hiding and binding.

- 1 $A \longrightarrow B$ (insecure channel): PK_A
- 2 B calculates \hat{h}_A as $h(\hat{PK}_A)$ and generates a syntactically-correct sentence from \hat{h}_A
 B displays this sentence on its screen.
- 3 A calculates h_A as $h(PK_A)$ and generates a syntactically-correct sentence from h_A
 A vocalizes this sentence using an audio speaker.
- 4 The user compares the sentence on B 's display with the vocalized sentence. If they are equal and the read out sentence is indeed originated from A the user accepts the protocol.

Figure 3.9: L&C type 3 one-way authentication protocol. B authenticates A .

3.14.5.4 Overall Pairing Time

According to [10], the average user can pair two devices using *L&C* in 23-32 seconds (depending on the protocol variant).

3.14.6 Human-Assisted Pure Audio Device Pairing

Human-Assisted Pure Audio Device Pairing (HAPADEP) proposed by Soriente *et al.* [3] uses audio channels to exchange both, public keys and hashes of public keys. In contrast to the protocols presented so far, HAPADEP does not need an additional in-band channel like Bluetooth or Wi-Fi. All data is transferred over audio channels.

3.14.6.1 Device Requirements

We assume that both devices are equipped with a speaker and a microphone to play and record audio sequences.

3.14.6.2 Protocol

The protocol is divided in a *transfer phase* and a *verification phase*. In the transfer phase both devices exchange their public keys using a fast, human unpleasant codec. In the verification phase, both devices present a human pleasant audio sequence (e.g. a melody) to the user. This audio sequence encodes a hash to the previous exchanged public keys. The user compares the two audio sequences and accepts the pairing, if and only if the audio sequences are equal. Further the user has to guarantee that the two melodies are indeed originated from A and B .

HAPADEP uses two different audio channels to exchange and verify public keys, because the user can not guarantee what the recoding device in the transfer phase exactly receives. The separate verification phase on the other hand gives a satisfying proof to the user whether the public key exchange was successful or not.

3.14.6.3 Protocol Assumptions

The attacker can not overlay the audio signal (melody) to fool the user into accepting two different integrity checksums.

3.15 Authentication by Integrity Checksum - OOB Transfer (P6)

Protocols in class P6 use an unspoofable OOB channel to transfer A 's integrity checksum to B . B compares the received checksum with his checksum. If they are equal he knows that the public key has sent by the device which sent the integrity checksum. The user notifies device A about the comparison result by pushing an accept/reject button on A 's device. A assumes that the received public key is authentic, if the accept button is pushed on his keyboard.

3.15.1 Channel Properties

The attacker can read from but not write on the OOB channel.

3.15.2 Secure Device Pairing based on a Visual Channel

Secure Device Pairing based on a Visual Channel (VIC) proposed by Saxena *et al.* [18] describes how mutual message authentication can be achieved with a one-way visual channel, where SiB (3.5.2) needs two visual channels, one in each direction.

3.15.2.1 Device Requirements

For using ephemeral public keys, B has to be equipped with a good enough camera to display 2D barcodes that encode at least 48 bits of data.

3.15.2.2 Protocol

The protocol is shown in Figure (3.10).

3.15.2.3 What the Protocol Achieves

At the end of a successful protocol run B knows that the public key $\hat{P}K_A$ was sent by the device that has been photographed. A knows that the public key $\hat{P}K_B$ was sent by the device that indicates *accept* to the user.

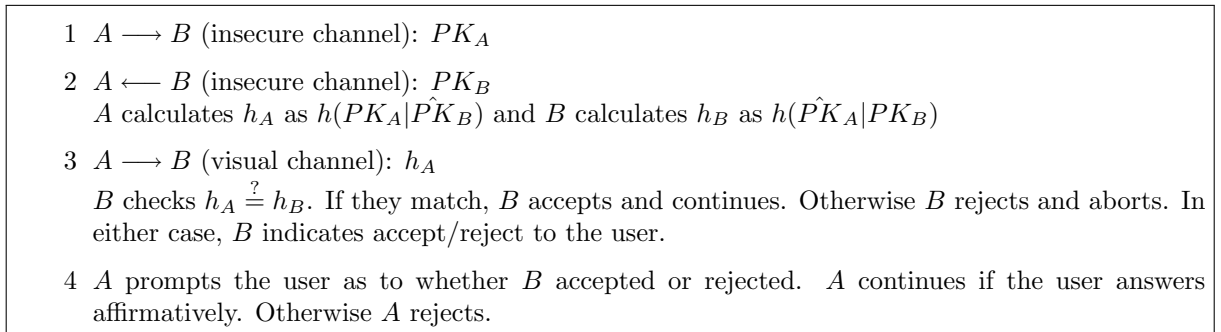


Figure 3.10: VIC mutual authentication protocol [18].

3.15.3 Integrity Codes

DH-IC proposed by Čagalj and Čapkun [24] is based on the short string comparison protocol DH-SC (3.8) but provides in addition a special OOB channel that ensures the integrity of a transmitted message. The protocol is shown in Figure (3.11). As in DH-SC the goal of the protocol is to exchange and authenticate Diffie-Hellman public keys (PK_A, PK_B) .

3.15.3.1 Attacker Model

We assume that the attacker can not disable the communication channel between A and B , e.g. by putting one device in a Faraday cage, but he can jam the transmission. In this case the receiver will still get the message from the sender, superimposed by the attacker's messages. We further assume that the attacker is computationally bounded.

3.15.3.2 Protocol

In DH-SC the user has to compare the two strings i_A and i_B to check the integrity of the transmitted public keys, where as in DH-IC this comparison is done by device B . Since we need an authentic copy of i_A on device B to do that, the protocol/user has to provide an authentic channel from A to B . As in VIC (3.15.2) we assume that the user pushes an accept/reject button on device A at the end of the protocol to inform A about the comparison result $i_A \stackrel{?}{=} i_B$.

3.15.3.3 Channel Properties and Coding

The OOB channel in DH-IC uses a coding called integrity code (I-code) that relies on (physical) channel properties to ensure the integrity of a transmitted message. We present here only a short overview of integrity codes. A detailed description can be found in [24].

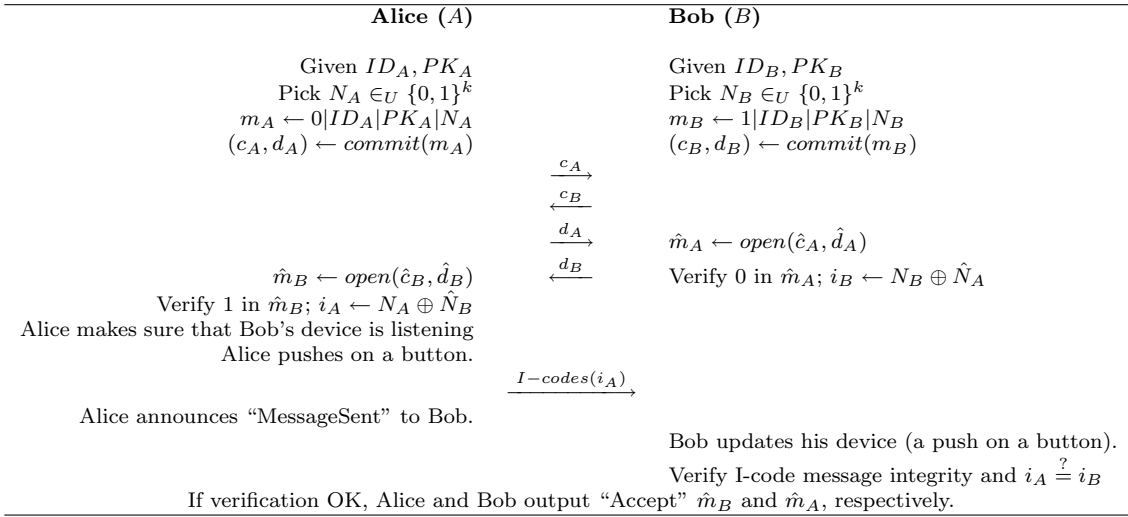


Figure 3.11: DH-SC key agreement enhanced with I-codes (DH-IC protocol) [4].

Integrity codes allow for the receiver to verify if a message has been changed during transmission. To do so, we use a special coding that relies on the channel property that a symbol "1" can neither be blocked nor changed to "0" in a communication sequence without being detected (except with a negligible probability). The idea is to encode every word to a binary sequence with a fixed number of "0" and "1" symbols in it. The decoding of a binary sequence aborts if the ratio between the number of "0" and "1" symbols is incorrect. An adversary can therefore only change a message by blocking or flipping at least one "1" in a communication sequence which we assume is prevented by the channel.

To ensure the above channel property in the context of radio transmission, we can profit from the fact that it is hard for an adversary to decrease the power of a signal below the background noise level, since this can only be done by sending out an inverted signal with exactly the same characteristics. Let $p_r(i)$ be the average power received in the interval corresponding to the i 'th symbol. The protocol defines two signal power levels p_0 and p_1 with $p_1 > p_0$ and p_0 background noise level. To transmit the i 'th symbol "1", the sender generates a signal with random characteristics but $p_r(i) \geq p_1$. To transmit the i 'th symbol "0" the sender generates a signal with $p_r(i) \leq p_1$. The demodulation on the receiver side works analogous by measuring the signal power. The random signal characteristics make it (even more) difficult for an adversary to send out an inverted signal and therefore to block or flip a symbol "1".

3.15.3.4 Assumptions

We assume that A is sending constantly at a given location and that B is listening on the correct channel during A 's transmission. We further assume that the distance between A and B is smaller than d , where d is the maximal distance where we can still guarantee that the power level of every symbol "1" at B is higher than or equal to the threshold power level

p_1 .

3.15.3.5 Authentication Through Presence

I-codes allows us to authenticate a public key solely by awareness of the presence of the sender. We can therefore use I-codes to realize the bank use case at the end of Chapter 2.

3.15.4 Distance Bounding

The protocol DH-DB proposed by Čagalj and Čapkun [24] is based on the short string comparison protocol DH-SC (3.8). The protocol guarantees that the exchanged Diffie-Hellman public keys are sent (or forwarded) by an entity that is within a determined integrity region. Figure (3.12) shows the integrity region for A and B . At the end of the protocol, both devices ask the user to check if no other radio device is within its integrity region. If this is the case, A and B can be sure that the public keys are not modified during transmission and that they are indeed sent by A , B respectively.

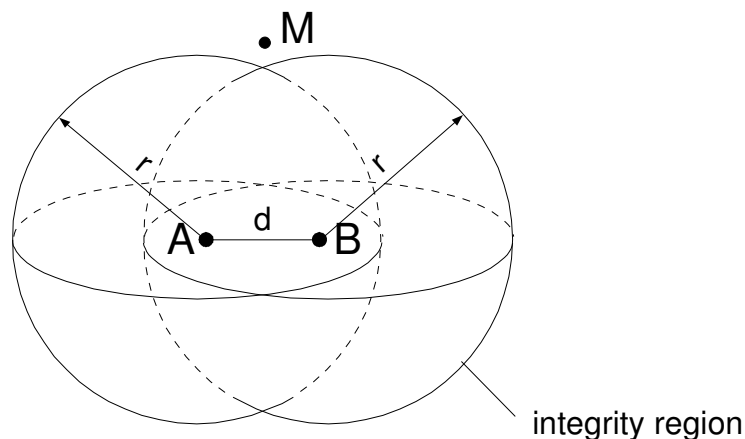


Figure 3.12: Integrity region of device A and B . d is the distance between A and B and r is the calculated upper bound on d .

3.15.4.1 Assumptions

The attacker can not be present in A 's and B 's integrity region during device pairing without being noticed by the user.

3.15.4.2 Protocol

The protocol is shown in Figure (3.13). A calculates the upper bound r on the distance to the other device by $r = (c \cdot t)/2$ where t is the average round-trip time of a challenge-response scheme and c the speed of the used signal type. The round-trip time is measured in the distance-bounding phase of the protocol. The signal speed depends on the transmission medium (usually air) and the signal type. For ultrasound we use $c = \text{speed of sound}$ and for radio $c = \text{speed of light}$.

3.15.4.3 Protocol Requirements

Both devices are equipped with hardware for fast bit transmission. For measuring the distance using ultrasound, both devices need to be equipped with speakers and microphones.

3.15.4.4 Location-Based Authentication

Distance-bounding protocols allow us to authenticate a message, based on the location of a sender. Such a protocol can therefore be used to realize the “restaurant-use-case” at the end of Chapter 2.

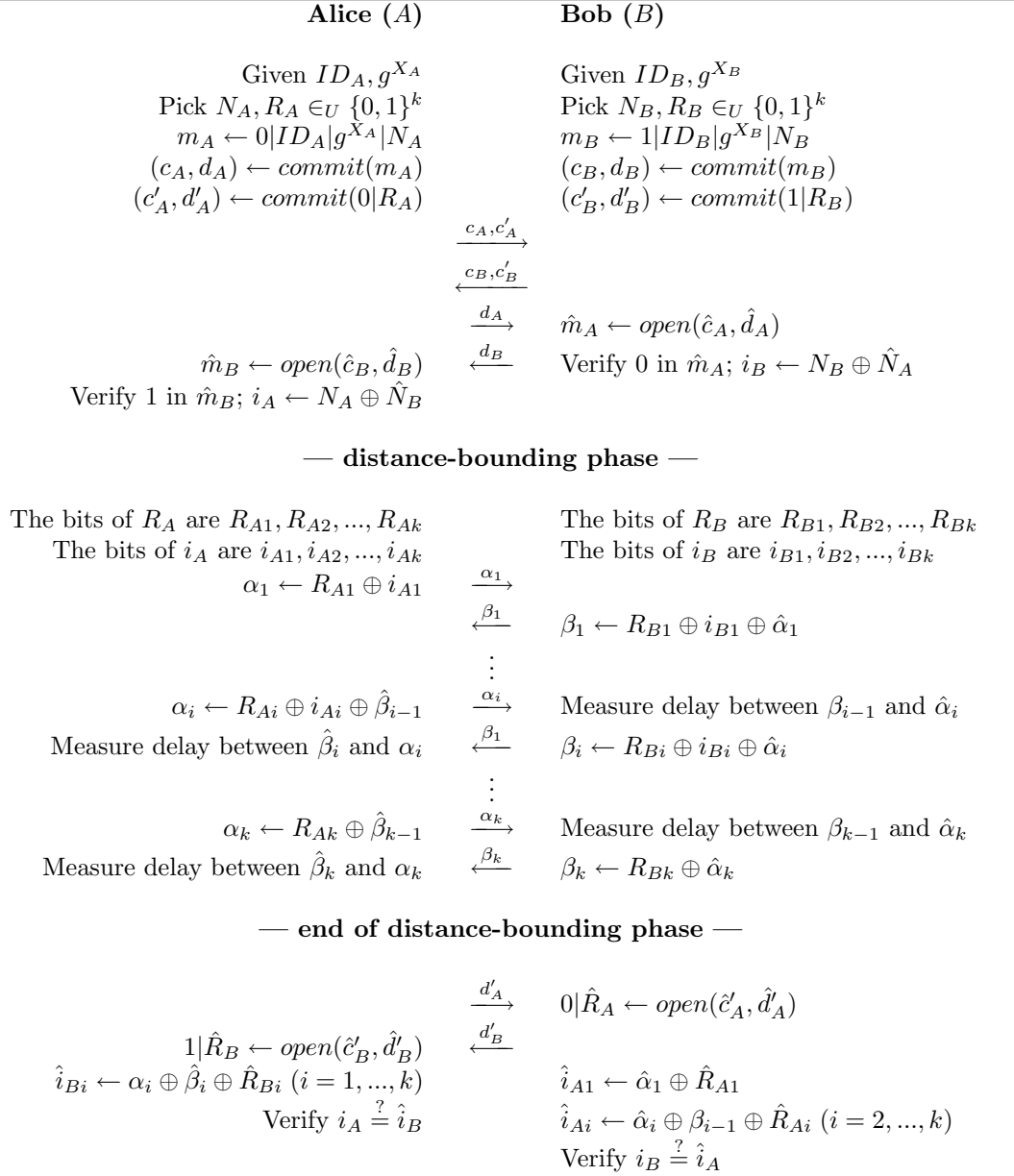
3.16 Authentication Using a Shared Secret (P7,P8)

Protocols in this section authenticate public keys using a short shared secret key. Figure (3.14) shows one round of such an authentication protocol, a variant of the MANA III protocol [9]. The short shared secret key is either generated by the user or by a device. If it is generated by a device A , either the user forwards it to B (P7) or A herself sends it to B over a secure OOB channel (P8). We consider here only protocols from class P7.

3.17 Authentication using a Shared Secret - User-Assisted (P7)

3.17.1 Button-Enabled Device Pairing

Button-Enabled Device Pairing (BEDA) proposed by Soriente *et al.* [20] uses a single button on A and/or B to generate a short secret key between A and B . The short secret key is then used to authenticate (Diffie-Hellman) public keys.



The users visually verify that there are no other users/devices in their “integrity region”.

Figure 3.13: DH-DB: DH key agreement based on distance-bounding [4].

- 1 A generates a long random value R_{iA} and computes the commitment
 $c_{i1} = \text{commit}(1, PK_A, \hat{PK}_B, P_i, R_{iA})$
 $A \rightarrow B: c_{iA}$
- 2 B generates a long random value R_{iB} and computes the commitment
 $c_{i2} = \text{commit}(2, PK_B, \hat{PK}_A, P_i, R_{iB})$
 $A \leftarrow B: c_{iB}$
- 3 A opens its commitment by sending R_{iA} to B
 $A \rightarrow B: R_{iA}$
- 4 B checks $\hat{c}_{iA} \stackrel{?}{=} \text{commit}(1, \hat{PK}_A, PK_B, P_i, R_{iA})$ and aborts if it does not hold.
- 5 B opens its commitment by sending R_{iB} to A
 $A \leftarrow B: R_{iB}$
- 6 A checks if $\hat{h}_{iB} \stackrel{?}{=} \text{commit}(2, \hat{PK}_B, PK_A, P_i, R_{iB})$ and aborts if it does not hold.

Figure 3.14: Round i of mutual authentication of the public keys PK_A and PK_B using a shared secret key P . The protocol steps will be repeated for every bit i of P [23].

3.17.1.1 Attacker Model

The attacker can not eavesdrop the secret key that is sent from the user to A and B or forwarded by the user from A to B . The attacker is computationally bounded.

3.17.1.2 Device Requirements

Device A needs a single button and device B either a button, a display, a LED or a vibrating motor.

3.17.1.3 Protocol

The paper presents four different protocol variants, called display-to-button (D-to-B), short vibration-to-button (SV-to-B), long vibration-to-button (LV-to-B) and button-to-button (B-to-B). The first 3 variants are shown in Figure (3.15). The only difference between these variants is the way device A presents the bits of the generated short secret P to the user. Each bit of P is encoded by A using the delay between two impulses. In D-to-B an impulse can be a blinking square on A 's display or a blinking LED. In SV-to-B an impulse is a short vibration and in LV-to-B an impulse is either the beginning or the end of a long vibration. For every impulse the user receives from device A , she has to press (or release) a single button at the same time on device B . Device B can now decode the bits of P using the time interval between two button events. We assume that these protocol variants realize a secure channel between A and B . By demonstrative identification of the sending/receiving

device the user guarantees that indeed A and B are communicating. To guarantee that only B can read from the channel the user additionally needs to verify that no adversary records the time interval between two impulses or button events (e.g. with a video camera).

- 1 A generates a short secret P and presents each bit of P sequentially to the user. The user forwards these bits to B using a single button on B .
- 2 $A \rightarrow B$ (insecure channel): PK_A
- 3 $A \leftarrow B$ (insecure channel): PK_B
- 4 A and B authenticate their respective public keys PK_A and PK_B using P . One round of this mutual authentication protocol is shown in Figure (3.14).

Figure 3.15: BEDA mutual authentication protocol (Variants D-to-B, SV-to-B and LV-to-B).

In the fourth variant of BEDA, called Button-to-Button (B-to-B), the user directly generates a short shared secret P on A and B by simultaneously pressing a button, multiple times, on both devices. A and B derive P from the random timing intervals between two button events and use P to authenticate their public keys. This protocol is shown in Figure (3.16). As in the three previous protocol variants we assume that the involved channels (from the user to both devices) are secure.

- 1 The user generates a short secret P on both devices A and B by simultaneously pressing a button multiple times on both devices.
- 2 $A \rightarrow B$ (insecure channel): PK_A
- 3 $A \leftarrow B$ (insecure channel): PK_B
- 4 A and B authenticate their respective public keys PK_A and PK_B using P . One round of this mutual authentication protocol is shown in figure 3.14.

Figure 3.16: BEDA mutual authentication protocol (Variant B-to-B)

3.17.1.4 What the Protocol Achieves

Under the assumption that the user correctly forwards the short term key from the device A and B , the protocol guarantees that the received public keys are indeed sent by A , B , respectively.

3.17.1.5 Overall Pairing Time

According to [20], the average user can pair two devices using BEDA in 53-73 seconds (depending on the protocol variant).

3.17.2 Shake Well Before Use (type 2)

Protocol type 1 of Shake Well Before Use is described in Section (3.9.2). In type 2, the accelerometer data is used to generate two secret feature vectors on A and B , which are then used to authenticate Diffie-Hellman public keys. Note that this protocol type uses a better known design than type 1, but is computationally more expensive and less dynamic.

3.17.2.1 What the Protocol Achieves

At the end of a successful protocol run, both devices guarantee, that the Diffie-Hellman public key they received was sent by the other shaken device.

3.17.3 Bluetooth Pairing

Bluetooth Pairing is standard to create a shared secret key on two Bluetooth devices. In Bluetooth version 1.x, this is done by entering a short secret passkey (e.g. 4-digit PIN) on both devices. In some very insecure implementations, one device (e.g. a headset) defines a fix passkey (e.g. 0000) which the user has to enter on the other device (e.g. mobile phone). The secret passkey is used to authenticate Diffie-Hellman public keys.

3.17.3.1 Device Requirements

Both devices need a keyboard.

3.17.3.2 What the Protocol Achieves

At the end of a successful protocol run, both devices, guarantee, that the public key they received was sent by an other device that knows the passkey.

3.18 Another Protocol Classification

Another classification for device pairing protocols is shown in Figure (3.17).

3.18.1 User Assisted Protocols

User-assisted device pairing protocols (protocol classes P3,P5,P7) are protocols where the user is the central element. Besides choosing the devices to pair, she either forwards a key from A to B (BEDA,FiB), compares two integrity checksums (DH-SC, L&C, HAPADEP) or enters a secret key on two devices (Bluetooth Simple Pairing).

User-Assisted	OOB Channel	Context-Based
BEDA	Cable/Infrared as OOB	Shake Well Before Use
Feeling-is-Believing	Seeing-is-Believing	DH-DB (location-based)
DH-SC	VIC	DH-IC (by presence)
Loud and Clear	Shake Them Up!	
HAPADEP	DH-IC	
Bluetooth Simple Pairing		

Figure 3.17: Protocol classification based on location/context/user-assistance

3.18.2 OOB Channel

Protocols in the second column of (3.17) provide an OOB channel with special security properties (e.g. a visual channel that guarantees the integrity of transmitted data).

3.18.3 Context-Based

In the third category, called context-based, are protocols that use certain aspects of a common device context to achieve a security goal. The common context for Shake Well Before Use, for example, is the users random movements. Distance-bounding protocols can be seen as a subclass of context based protocols that use the location of a device to achieve a security goal.

Note that a protocol can be in more than one category. DH-IC for example provides an OOB that ensures the integrity of a transmitted message. On the other hand it enables *authentication through presence* what can be seen as a context-based property.

You can't consider the problem of defense without first understanding the problem of attack.

(Doug Tygar)

4 Attacks

4.1 Compromising Devices

An adversary may attempt to attack a device by installing malicious software on the device. This could allow him to read out a secret key or to replace a public key in a memory. Moreover if the device is compromised before device pairing, we can not guarantee that the protocol will be executed correctly. Such attacks can not be prevented by device pairing protocols. It is the task of the user and the device to ensure that a device can not be compromised, e.g. by implementing a good system policy. I do not consider such attacks or prevention mechanisms here.

4.2 Denial-of-Service Attacks

In the context of device pairing, the goal of a *Denial-of-Service* (DoS) attack is, to preclude A and B from creating a security relation. Most device pairing protocols are vulnerable to DoS attacks, but they differ in how easy the user/device can detect such an attack. If the user detects a DoS attack and if she can determine the attack source, then she could try to eliminate it, or failing this, change the environment. I consider only the most important DoS attacks for the protocols presented in Chapter 3.

General Jamming radio or infrared channels.

Shake-Them-Up! Insert a bogus packet with source address A or B . A and B abort at the end of the protocol when they notice, that they have not agreed to the same keys. An extended version of the protocol can partially resist against DoS attacks [6]. To do this the receiving device acknowledges a packet by sending the same packet back to the sender. A protocol round with more than two packets is simply ignored. This protocol needs twice as many packets as the original version to generate a key of the same length.

Integrity Codes Insert a bogus packet with symbol "0" or "1". The decoding of at least one codeword will fail. Depending on the implementation of integrity codes the sender has to begin again with the whole transmission or need only to resend the corrupt codeword.

HAPADEP Play a loud sound (e.g. tone or noise) to prevent the receiving device from recording what the sending device is playing. This attack can be recognized (actually,

heard) by the user if the sound is in the audible range. To prevent DoS attacks with sounds outside the audible range, the devices could filter out all frequencies unused for data transmission.

Loud and Clear Same as for HAPADEP. But here, the user is prevented from recognizing the audio sequence(s).

4.3 Man-in-the-Middle Attacks

Man-in-the-Middle (MitM) attacks are the most widespread attacks against device pairing protocols. In a MitM attack an adversary impersonates A when he communicates with B and B when he communicates with A . I state here a few examples for MitM attacks in the context of device pairing.

4.3.1 MitM Attack against Diffie-Hellman Key Agreement

The Diffie-Hellman key agreement protocol (5.9) solves the problem of establishing a security relation $A \bullet \text{---} \bullet B$ if no secret channels between A and B are available. Unfortunately the protocol is vulnerable to a MitM attack. The attack is shown in Figure (4.1). At the end of the attack, the adversary M shares a key k_A with A and a key k_B with B .

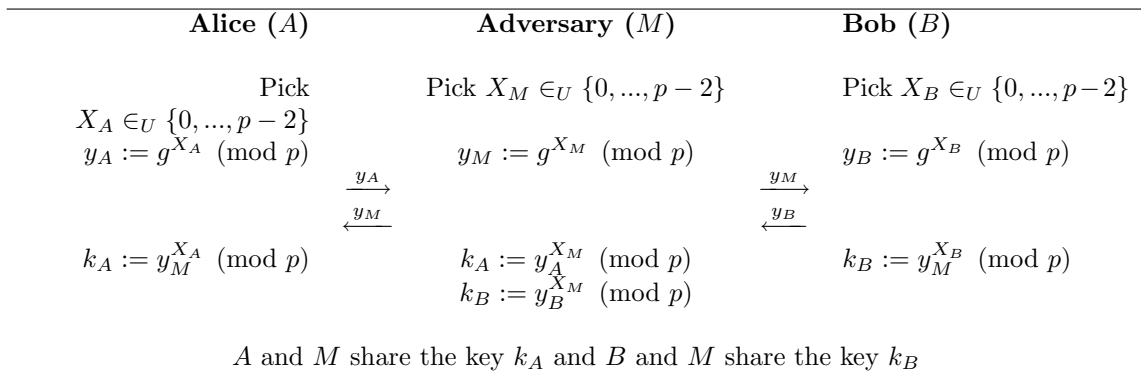


Figure 4.1: MitM attack against Diffie-Hellman key agreement protocol. The prime p and the generator g are publicly known parameters.

4.3.1.1 Conclusion

We must authenticate the public keys y_A and y_B .

4.3.1.2 MitM Attacks in Practice

In the last section we saw a protocol that is vulnerable to MitM attacks. But are such attacks in today's communication networks even possible? The answer is yes. I state here one example of a concrete MitM attack in networks using ARP (Address Resolution Protocol).

Assume a user wants to pair his two handhelds A and B by creating a shared secret key using Diffie-Hellman key agreement. A and B are enrolled into an insecure WLAN (no WEP, WPA etc.). The adversary has access with his laptop (M) to this WLAN. Since A and B are in the same network segment both devices directly communicate with each other by sending packets to the other device hardware (MAC) address (note that this attack is also possible when A and B are not in the same network segment). The adversary now sends fake ARP messages to A and B , so that both devices bind the IP address of the other device to the hardware address of M . All IP packets exchanged between A to B are now transferred over M . This kind of attack is called *arp spoofing* [12].

4.3.1.3 Conclusion

It is relatively easy to perform a MitM attack in unprotected wireless network standards.

4.3.2 MitM Attack Against Shake Them Up!

As a third MitM example we consider the protocol Shake Them Up! (3.10.2) using an anonymous broadcast channel to agree A and B to a shared secret key. To transmit a secret bit from A to B or vice versa, both devices need to know the others device address. Remember that a bit is encoded in the source field of a packet and that the device addresses are exchanged in the first two start messages of the protocol. An adversary who can fake these two addresses can launch a MitM attack and decode the transferred secret bit. Assume the adversary fools A and B to believe that the other device address is $M(A)$, $M(B)$ respectively. Thus he knows that each packet with a source field set to A or $M(B)$ is sent by A . If the field is set to A the packet represents 1. If it is set to $M(B)$ it represents 0. In the same way he can decode the packets from B . To complete the MitM attack, he replaces $M(B)$ with B and $M(A)$ with A in the source field of each packet. The generated key on A and B is now also known to the adversary without being noticed by A and B .

4.3.2.1 Conclusion

We must authenticate the two start messages.

4.4 Bluetooth Pairing Attack

The security of simple Bluetooth pairing relies on the user's choice of a secret PIN, which is often much too short. Yaniv Shaked and Avishai Wool showed that a passive attacker can derive a 4-digit PIN from an eavesdropped pairing communication in less than 0.06 seconds on a common PC using brute force algorithm [19].

4.4.1 Conclusions

Standard Bluetooth pairing using a short PIN is insecure in an hostile environment. Users should therefore refrain from entering the PIN for pairing the Bluetooth devices as much as possible. As PINs length ranges from 8 to 128 bits, it is also advisable to choose the PIN as long as possible.

The Bluetooth Special Interest Group (SIG) reacted to these concerns by creating Secure Simple Pairing (Bluetooth version 2.1). The protocol supports four association modes: Passkey entry, Numeric Comparison, 'Just Works' and an OOB model (particular with Near Field Communication technology). For a detailed description of these modes I refer to the paper [11].

4.5 Reflection Attack

Consider protocol DH-SC (3.8) where A would calculate message m_A as $ID_A|g^{X_A}|N_A$ and B m_B as $ID_B|g^{X_B}|N_B$. Then an adversary can fool A and B into accepting their own public keys by reflecting every message sent from A back to A and every message sent from B back to B . In this case both devices calculate the same checksum $i_A = N_A \oplus N_A = i_B = N_B \oplus N_B = 0^k$. To prevent such an attack A adds 0 to the beginning of m_A and B (the answering device) adds 1 to the beginning of m_B . Both devices can now detect a reflection attack by verifying this first character in the received message m_A, m_B respectively.

4.6 Attacks against Distance-Bounding Protocols

A distance-bounding protocol can for example be used by an automatic door system, to verify that a visitor possesses a secret key and that he/she is within a predefined distance to the door. Only when both conditions hold, the door opens. In contrast to this example, we can profit from the presence of the user in a device pairing protocol. In DH-DB for example, the user verifies that no other devices are within a calculated distance to A and B .

We consider a user of device A that wants to verify that a public key Pk_B is indeed sent from device B . To do this A determines an upper bound u on the (euclidean) distance d to B . The user verifies that B is the only device that is d away from A . If this is the case she

knows that B must be the sender of Pk_B . The adversary M now tries to impersonate B and fool A into accepting his own public key. Since M does not want to be detected by the user, he has to be more than u away from A . We now consider A 's approaches to determine the correct upper bound to B (respectively M) and M 's attempt to fool A into accepting his public key as the public key of B .

Approach 1 A sends a challenge to B and waits for B 's response. From the response signal she measures the signal power and tries to derive the distance to B .

Attack 1 M could have special equipment to generate a strong signal and therefore to fool A that he is closer than he actually is.

Conclusion 1 In the context of device pairing, signal power analysis is not reliable to determine the distance to a device.

Approach 2 A measures the time between sending a challenge and receiving the answer. She then calculates the upper bound u from this time interval and the signal speed.

Attack 2 M can try to send the response before he gets the challenge.

Conclusion 2 The response must depend on the challenge and the challenge should not be predictable.

Approach 3 A uses ultrasound signals to measure the distance d .

Attack 3 M installs a small device S near A that forwards each message of A to M over a radio channel. If S receives a message from M it converts it back over an ultrasound channel to A . M can now respond to a challenge of A as he would be in the integrity region of A . A calculates an incorrect upper bound u on the distance to M because she assumes that the signal can not be faster than the speed of sound (radio signals are about 880'000 times faster than ultrasound signals in air¹).

Conclusion 3 If the user correctly checks that no other device is u away from B , then an adversary can not install device S near enough to A . If A uses radio signals to measure the distance to B , then M can not accelerate the signal speed using a device S , because we assume that information can not be transmitted faster than with the speed of light.

Approach 4 B first sends his public key over an insecure channel to A and then performs the distance measuring phase with A .

Attack 4 M replaces B 's public key with his own public key on the insecure channel and let B perform the distance measuring phase with A .

Conclusion 4 The distance measuring phase must depend on the public key, such that the sender of the public key must also be the entity performing the distance measuring protocol with A .

This is a set of *distance fraud attacks*, where M pretends to be nearer to A than he actually

¹Speed of radio signals \approx speed of light \approx 300'000 km/s, Speed of ultrasound in air \approx 340 m/s

is. The attacks are partially based on ideas from *mafia fraud attacks* and *terrorist fraud attacks* against identification and authentication systems [2, 16, 17].

4.7 Another Attacks Against Shake Them Up!

Shake Them Up! (3.10.2) is vulnerable to an active attacker if the following assumption does not hold: “*A* or *B* discard the generated n bit key if they have not both contributed at least, e.g. $n/3$ bits to the secret key”². Consider again the protocol example of Shake Them Up! already introduced in Chapter 3. (a copy of the protocol example is shown below). An adversary could drop all packets sent by *A* and *B* and insert arbitrary bogus packets $B_i, i = 1, \dots, n$ with source field set to *A* or *B*. *A* would then agree on the key $K_A = K_{A1}, K_{A2}, \dots, K_{An}$, where $K_{Ai} = 1$ if the i 'th packet she receives has a source field *B* and 0 if the i 'th packet has a source field *A*. *B* would agree on the key $K_B = \bar{K}_{A1}, \bar{K}_{A2}, \dots, \bar{K}_{An}$, where $\bar{1} = 0$ and $\bar{0} = 1$. In the last two steps the adversary broadcasts the values $a = \text{hash}(A|B|K_B)$ and $b = \text{hash}(B|A|K_A)$ such that both *A* and *B* accept the protocol run.

To prevent this attack, both devices should contributed enough bits to the generated key, so that the adversary can not generate the last two key validation values $a = \text{hash}(A|B|key)$ and $b = \text{hash}(B|A|key)$.

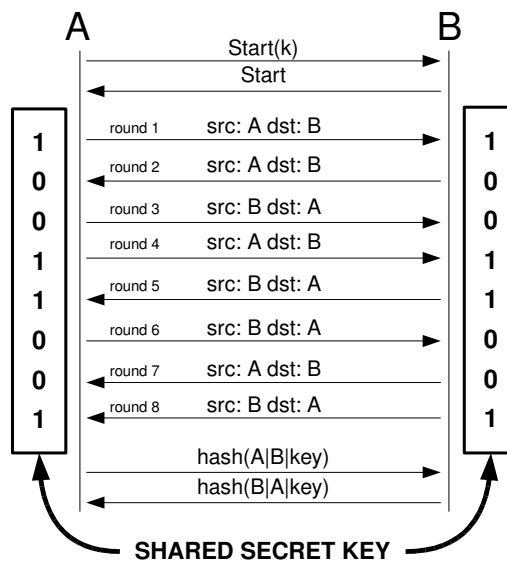


Figure 4.2: Key agreement protocol for movement-based pairing [6].

²I have not found this assumption in [6].

You can't defend. You can't prevent.
The only thing you can do is detect
and respond.

(Bruce Schneier)

5 Security Properties

In this chapter I describe the protocol classes, introduced in chapter 3, as channel and security relation transformations. An overview of these transformations is shown in the following figure, where I assume that insecure channels are available at any time.

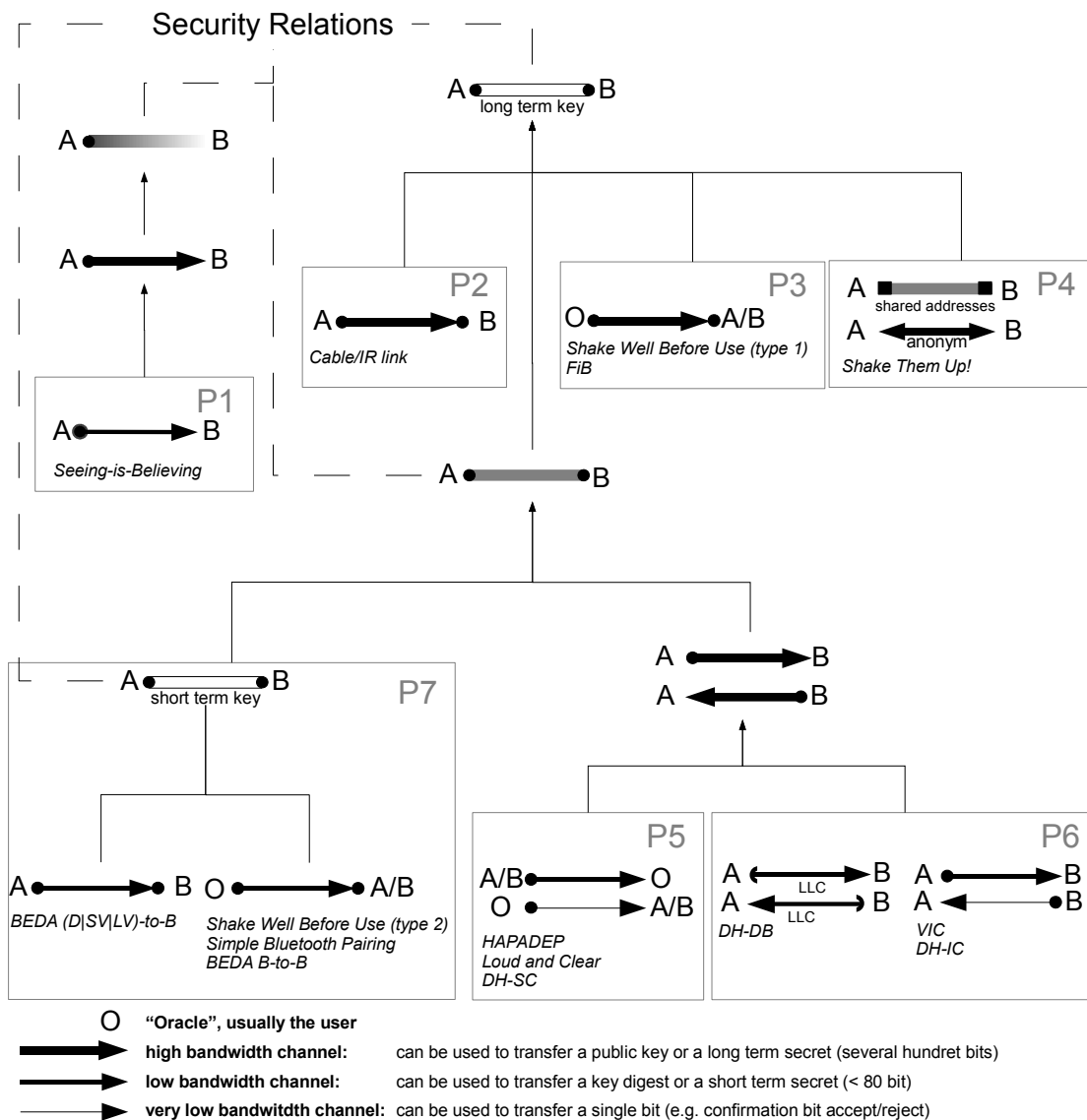


Figure 5.1: Overview of channel and security relation transformations

5.1 Terms and Notations

Inspired by the \bullet -calculus by U. Maurer and E.Schmid [13] I use the following channel notations. The channel properties are described in (1.4.1).

$A \longrightarrow B$ Insecure channel from A to B .

$A \bullet \longrightarrow B$ Authentic or authenticated channel from A to B .

$A \bullet \longleftrightarrow \bullet B$ Both-way secure channel between A and B .

I use the term *authentic channel* to refer to a channel providing authenticity by physical properties and/or user assistance and the term *authenticated channel* to refer to a channel, authenticated by a (cryptographic) protocol. Further note that the security properties of the channels are guaranteed by the user *and* the protocol, e.g. the channel guarantees integrity and the user guarantees that the correct device is sending.

5.1.1 Timing Aspects

Two time parameters t_i, t_j $i, j \in \mathbb{N}$ above a channel symbol, e.g. $A \bullet \xrightarrow{t_1} \xrightarrow{t_2} \bullet B$, denote when a messages must be sent (t_1) and when it arrives (t_2). This implies that $t_2 > t_1$. If the delay of a channel is not important, only one time parameter t or t_i appears above the channel symbol. $A \bullet \xrightarrow{t_3} \bullet B$ states therefore for a secure channel available at time t_3 .




Applied to shared secrets and authentic copies of public keys, $A \bullet \xrightarrow{t} \bullet B$ stands for a shared secret between A and B available at time $t' > t$ and $A \xrightarrow{t} \bullet B$ describes that A has an authentic copy of B 's public key at time $t' > t$.

5.1.2 Mutual Access to Channel Endpoints

We use the notation A/B to refer that both A and B have exclusive access to one end of the channel, e.g. $A/B \bullet \xrightarrow{t} O$ denotes an authentic channel available at time t where both A and B can send a message to O . O receives two messages at time t and knows for both messages whether the message was sent by A or B .

5.1.3 Channel Bandwidth

The line thickness of a channel symbol describes the bandwidth of the channel:

-  very low bandwidth channel, suitable to transmit 1 bit (e.g. ok/nok commitment).
-  low bandwidth channel, suitable to transmit a key digest or a short term key (< 80 bit).
-  high bandwidth channel, suitable to transmit a public key or a long term key (few hundred bits).

Note that one could think of using a low bandwidth channel to transmit a public key or a long term key by waiting longer for transmission completion. Since for almost all low bandwidth channels the user is involved in either way, this technique would not be user-friendly and is therefore inapplicable.

5.1.4 Channel Transformations

We use the following notation for channel or security relation transformations: the required conditions and primitives are on the left side, followed by \rightsquigarrow , and the derived primitives are on the right side.

We further use the following trivial transformations:

A one-way, high bandwidth and secure channel can be used to transmit a long term secret key.

$$A \xrightarrow{t} B \} \rightsquigarrow A \xrightarrow{t} B \quad (5.1)$$

A one-way, high bandwidth, authentic (or authenticated) channel can be used to transmit a public key.

$$A \xrightarrow{t} B \} \rightsquigarrow A \xrightarrow{t} B \quad (5.2)$$

5.2 OOB Transfer of Key Commitments (P1)

Devices in protocol class P1 provide a (physically) low bandwidth OOB channel $\bullet \longrightarrow$ to authenticate a high bandwidth, insecure channel \longrightarrow . This is depicted by the following channel transformation:

$$\left. \begin{array}{l} A \xrightarrow{t_1} B \\ A \xrightarrow{t_3} B \\ t_2 < t_4 \end{array} \right\} \rightsquigarrow A \xrightarrow{t_3} B \quad (5.3)$$

To send a message m over $\xrightarrow{t_3} B$, A first sends a commitment $c = h(m)$ to m over the authentic channel $\xrightarrow{t_1} B$ to B . Afterwards, A sends m over the insecure channel $\xrightarrow{t_3} B$ to B . B can now use c to verify the authenticity of m (by checking $c \stackrel{?}{=} h(\hat{m})$).

By composing transformation (5.2) and (5.3) we get the transformation:

$$\left. \begin{array}{l} A \xrightarrow{t_1} B \\ A \xrightarrow{t_3} B \\ t_2 < t_4 \end{array} \right\} \rightsquigarrow A \xrightarrow{t_4} B \quad (5.4)$$

5.2.1 Seeing-is-Believing

Seeing-is-Believing (3.5.2) is an application of transformation (5.4). The visual OOB channel ensures the integrity of the transmitted public key and the user makes sure that indeed A is sending. This provides the necessary channel $A \xrightarrow{t_1} B$.

5.3 OOB Key Transfer (P2)

Protocols of class P2 generate a security relation $A \bullet \longrightarrow B$ by transferring a long term secret key from one device to the other, using a (physically) secure channel. This trivial transformation is stated in (5.1).

5.3.1 Shielded Cable/ Infrared Link

A shielded cable or an infrared link can be used to realize a high bandwidth, secure channel $\bullet \longrightarrow \bullet$ necessary for transformation (5.1). For a cable connection, we assume that the transmission can not be overheard or modified by an adversary without being detected by

the user. For an infrared link we assume that the user ensures that no other infrared device is in line of sight (to prevent MitM attacks). By demonstrative identification of the devices at the endpoints of the cable respectively in line of sight, the user ensures that only A can send and only B can read from the channel.

5.4 User-Assisted Key Agreement (P3)

Devices in class P3 have exclusive access to a random oracle (usually the user) providing enough random data to create a long term key $A \longleftrightarrow B$. I use the notation $O \xrightarrow{t} A/B$ to show that A and B get the same random data from the oracle O . This leads to transformation:

$$O \xrightarrow{t} A/B \} \rightsquigarrow A \xrightarrow{t} B \quad (5.5)$$

5.4.1 Shake Well Before Use (type 1)

In type 1 of Shake Well Before Use (3.9.2), the user provides the secure channel $\bullet \longrightarrow \bullet$ to both devices, by shaking the devices in one hand. I assume here that an adversary is unable to derive the generated accelerator data (secrecy channel property), e.g. by a side channel attack and that he can not influence the accelerator data, without being noticed by the user (authenticity channel property).

5.5 OOB Key Exchange (P4)

Devices in class P4 provide two high bandwidth, secret OOB channels $A \longrightarrow B$ and $A \longleftarrow B$ which we abstract here as a both-way secure channel $A \longleftrightarrow B$. The devices agree to a shared secret key by exchanging secret key material over these channels. Note that this protocol class differs from protocol class P2 where one device generates the whole key and sends it over a secure channel to the other device.

$$A \longleftrightarrow B \} \rightsquigarrow A \xrightarrow{t} B \quad (5.6)$$

5.5.1 Shake Them Up!

Shake Them Up! (3.10.2) provides together with the user an anonymous broadcast channel denoted as $A \xrightarrow{\text{anym}} B$. This channel ensures that nobody except A and B can determine from which particular sender a signal originates. Beside this channel, both devices need an

authentic copy of the others device address¹. This is denoted by the symbol $A \xrightarrow{\text{auth}} B$. Note that this is not a security relation and has nothing to do with keys. An adversary M who can fool at least one device A or B into accepting a wrong device address (e.g $A \xrightarrow{\text{auth}} M(B)$, where $M(B)$ denotes M masquerading as B), can launch a MitM attack against the protocol (as we have seen in 4.3.2). $A \xrightarrow{\text{auth}} B$ can be created by comparing A and B 's addresses displayed on both devices or by means of techniques from other device pairing protocols. Having these two primitives, the protocol provides a both-way secure channel:

$$\left. \begin{array}{l} A \xrightarrow{t_1} B \\ A \xleftrightarrow[\text{anym}]{} B \\ t_1 < t_2 \end{array} \right\} \rightsquigarrow A \xleftrightarrow{t_2} B \quad (5.7)$$

The last two messages $a = \text{hash}(A|B|\text{key})$ and $b = \text{hash}(B|A|\text{key})$ of the protocol are used to validate the exchanged key (key). We assume that $\text{hash}()$ is a preimage resistant hash function, to ensure that the adversary can not learn the key from the hash digest. As we have seen by attack (4.7), the adversary can insert bogus bits into the key, during key agreement. This would violate our assumption that the channel between A and B provides authenticity. By requiring that A and B abort the protocol run if they have not contribute at least, e.g. $n/3$ bits to the generated n bit key, we ensure that the adversary can not bring A or B into accepting the protocol run. This leads to that an adversary can not insert or modify packets send on the anonymous broadcast channel.

By combining transformation (5.7) and (5.6) we get:

$$\left. \begin{array}{l} A \xrightarrow{t_1} B \\ A \xleftrightarrow[\text{anym}]{} B \\ t_1 < t_2 \end{array} \right\} \rightsquigarrow A \xleftrightarrow{t_2} B \quad (5.8)$$

5.6 Diffie-Hellman Key Agreement (P5-P8)

Diffie-Hellman Key Agreement allows us to generate a security relation $A \xleftrightarrow{\text{key}} B$ if we can generate a security relation $A \xrightarrow{\text{auth}} B$, i.e. if we can authenticate Diffie-Hellman half-keys. This is depicted by the following security relation transformation:

$$A \xrightarrow{\text{auth}} B \} \rightsquigarrow A \xleftrightarrow{\text{key}} B \quad (5.9)$$

¹It is sufficient to have an authentic copy of any value that can be used in the source field of a packet.

For the last of this chapter we consider security transformations to generate a security relation $A \bullet \text{---} \bullet B$ and therefore, due to transformation (5.9), to generate a shared secret key $A \bullet \text{---} \bullet B$.

5.7 User-Assisted Key Agreement (P5)

Protocols in class P5 need the user as a comparison oracle to verify if two strings (*integrity checksums*) are equal. If they are equal, the protocol guarantees that previously exchanged public keys (e.g. Diffie-Hellman half-keys) are authentic. Channel transformation (5.10) shows the involved channels for user-assisted protocols, based on integrity checking (class P5).

We assume that O is a comparison oracle (usually the user) with the following properties:

- If O gets two identical values at time t_5 from the channel $A/B \bullet \xrightarrow{t_5} O$ it sends “accept” on the channel $O \bullet \xrightarrow{t_6} A/B$.
- If O gets two different values at time t_5 from the channel $A/B \bullet \xrightarrow{t_5} O$ it sends “reject” on the channel $O \bullet \xrightarrow{t_6} A/B$.

$$\left. \begin{array}{l}
 A \xrightarrow{t_1} B \\
 A \xleftarrow{t_4} B \\
 A/B \bullet \xrightarrow{t_5} O \\
 A/B \xleftarrow{t_6} O \\
 t_5 > t_2, t_4 \\
 t_6 > t_5
 \end{array} \right\} \rightsquigarrow \begin{array}{l}
 A \bullet \xrightarrow{t_6} B \\
 A \xleftarrow{t_3} B
 \end{array} \quad (5.10)$$

The first two channels $A \xrightarrow{t_1} B$ and $A \xleftarrow{t_4} B$ are used to exchange public keys and calculate the integrity checksum. After that (at time t_2, t_4), both devices send the integrity checksum over $\bullet \xrightarrow{t_5}$ to O . O processes as describe above. A and B accept at time t_6 the received public key if and only if they receive *accept* from O . The insecure channels are therefore authenticated at time t_6 depicted as $A \bullet \xrightarrow{t_6} B$ and $A \xleftarrow{t_3} B$.

By applying transformation (5.2) to both high bandwidth, authenticated channel from (5.10) we get the necessary security relation $A \bullet \text{---} \bullet B$ for protocols in class P5:

$$\left. \begin{array}{l} A \xrightarrow{t_1} B \\ A \xleftarrow{t_3} B \end{array} \right\} \rightsquigarrow A \bullet \text{---} \bullet B \quad (5.11)$$

For all of implementations of class P5 protocols, the user selects A and B by demonstrative identification. The devices send the integrity checksum over a visual or audio channel to the user. Visual means that a device shows the integrity checksum on its display. Audio means that a device encodes the string to an audio sequence and plays the sequence using an audio speaker. For both channels the user checks that only A or B can send on the channel and that the message is not modified during transmission. This ensures the authenticity of the channel. The result of the comparison can be transmitted to the devices by pushing an accept/reject button on the device keyboards. The user ensures that she has exclusive access to the device keyboard during device pairing. This guarantees that nobody else can send on the channel $O \xleftarrow{t_6} \bullet A/B$.

5.8 Authentication by Integrity Checksum - OOB Transfer (P6)

Device pairing protocols in class P6 provide two low bandwidth authentic channels from A to B and vice versa to check the integrity of previous exchanged public keys.

5.8.1 Distance Bounding

We do not directly model the channel transformation for DH-DB here, because the protocol needs $6 + 2k$ channels, where k is a security parameter of the protocol. We describe instead an abstract version of a distance bounding device pairing protocol with the following channel transformation (5.12) and explain why DH-DB uses more channels in practice.

$$\left. \begin{array}{l} A \xleftarrow[r]{t_1} B \\ A \xrightarrow{t_2} B \\ t_1 < t_3 \end{array} \right\} \rightsquigarrow A \xleftarrow[r]{t_2} B \quad (5.12)$$

The channel $A \xleftarrow[r]{t_2} B$ stands here for a location limited (wireless) channel (LLC) where B knows that the sender of a message must be closer than r . r is therefore an upper bound on the actual (euclidean) distance between A and B . As we already have seen, such a LLC

can be realized by a distance-bounding protocol.

The insecure channel $\xrightarrow[t_2]{t_3}$ is used to transfer A 's public key to B . By sending a hash of this public key over $\xleftarrow[r]{t_3}$ we get the channel $\xleftarrow[r]{t_2}$.

A distance-bounding device pairing protocol allows us therefore to transfer the distance property of $\xleftarrow[r]{t_3}$ to a high bandwidth insecure channel $\xrightarrow{t_3}$. This is similar to transferring a \bullet from a low bandwidth to a high bandwidth channel (e.g. as in 5.3).

By verifying that only A is in the integrity region (3.12) of B during device pairing, the user knows that the public key received by B must be sent by A . This is depicted by the following transformation:

$$\left. \begin{array}{l} A \xleftarrow[r]{t_2} B \\ \text{User verifies integrity region of B} \\ \text{in the time between } t_2 \text{ and } t_3. \end{array} \right\} \rightsquigarrow A \xrightarrow[t_2]{t_3} B \quad (5.13)$$

5.8.1.1 DH-DB

DH-DB (3.13) uses a k -round distance-bounding phase, where k is the size of the two integrity checksums i_A and i_B . In each round only one bit is sent from A to B and B to A . This allows to calculate a tighter upper bound on the distance to the other communication partner because the processing delay on A and B can be very short. The challenges of A are calculated as $\alpha_i = R_{Ai} \oplus i_{Ai} \oplus \hat{\beta}_{i-1}$ $i \in 1, \dots, k$. R_{Ai} is a random bit, generated by A , that ensures that B can only predict the challenge with a probability of $1/2$. i_{Ai} is the i 'th bit of A 's integrity checksum i_A that B needs to compare with this integrity checksum i_B . The distance-bounding phase therefore depends on the integrity checksums of A and B . This ensures that the same entity performing the distance-bounding phase must have sent the public key. The last value $\hat{\beta}_{i-1}$ is the challenge of B received in round $i - 1$. A 's challenge bit α_i is therefore also a response the the previous challenge bit of B .

5.8.2 User-Assisted Feedback Channel

Both-way public key authentication can also be achieved when A and B only provide a one-way, low bandwidth OOB channel $\bullet \xrightarrow{t_3}$, e.g. from A to B . This is depicted by the following transformation:

$$\left. \begin{array}{l}
 A \xrightarrow{t_1} B \\
 A \xleftarrow{t_3} B \\
 A \xrightarrow{t_5} B \\
 A \xleftarrow{t_7} B \\
 t_5 > t_4 \\
 t_7 > t_2, t_6
 \end{array} \right\} \rightsquigarrow \begin{array}{l}
 A \xrightarrow{t_1} B \\
 A \xleftarrow{t_3} B
 \end{array} \quad (5.14)$$

The two insecure channels are used to exchange public keys. A and B calculate then an integrity checksums from the keys. If the checksum on A is equal to the checksum on B the protocol guarantees that the public keys are sent by A and B respectively. The low bandwidth authentic channel $A \xrightarrow{t_5} B$ is used to transmit A 's checksum to B . B can then compare the two checksums. To inform A whether the checksums are equal or not, the user forwards the result of the comparison from B to A by pushing an accept/reject button on A 's keyboard. This is depicted by the channel $A \xleftarrow{t_7} B$.

5.8.3 VIC

In VIC (3.15.2), the integrity checksum is a hash of the concatenation of A 's and B 's public key (3.15.2). The channel $A \xrightarrow{t_5} B$ is realized in the same way as in SiB, where the user takes a picture of A 's display with B 's camera. The authenticity of the channels $A \xrightarrow{t_5} B$ and $A \xleftarrow{t_7} B$ is guaranteed by the visual channel and the user by demonstrative identification of A and B .

To break the authentication scheme of VIC, the adversary has to find two values X, Y such that $h(K_A|X) = h(Y|K_B)$. Note that finding a collision in $h()$ is not enough since K_A and K_B are fixed.

5.8.4 Integrity Codes (DH-IC)

DH-IC extends DH-SC by the channel $A \xrightarrow{t_5} B$. Section (3.15.3) describes how the user and the protocol ensures the authenticity of a transmitted message sent over $A \xrightarrow{t_5} B$.

5.9 Authentication using a Shared Secret Key (P7,P8)

Protocols in class P7 and P8 generate a short term key $A \xrightarrow{\text{short}} B$ on both devices to authenticate Diffie-Hellman half-key afterwards.

Authentication by a short term key $A \xrightarrow{t} B$ is depicted by the following transformation.

$$\left. \begin{array}{l} A \xrightarrow{t_1} B \\ A \xleftrightarrow{t_2} B \\ t_2 > t_1 \end{array} \right\} \rightsquigarrow A \xrightarrow{t_2} B \quad (5.15)$$

A protocol for mutual authentication using a short shared secret was introduced in (3.16). Note that this protocol needs l insecure channels between A and B , where l is the size of the short term key $A \xrightarrow{t_1} B$.

5.9.1 BEDA (D|SV|LV)-to-B

In the BEDA protocol variants D-to-B, SV-to-B and LV-to-B (3.17.1) the user provides a one-way, low bandwidth secure channel $A \xrightarrow{\text{secure}} B$, by forwarding a message bit by bit from A to B . This is shown in channel transformation (5.16). Each channel on the left side is used to transmit one bit. The secrecy property of the channel is guaranteed by choosing an environment where the adversary has no visual access to A 's display and where no equipment is installed (e.g. camera, microphone) that can record the timing property of the button/vibration events. The user ensures that indeed A sends and B reads from the channel by demonstrative identification of the sending/receiving device.

$$\left. \begin{array}{l} A \xrightarrow{t_1} B \quad User \\ User \xrightarrow{t_3} B \quad B \\ \vdots \\ A \xrightarrow{t_{n-3}} B \quad User \\ User \xrightarrow{t_{n-1}} B \quad B \\ t_n > t_{n-1} > \dots > t_2 > t_1 \end{array} \right\} \rightsquigarrow A \xrightarrow{t_1} B \quad (5.16)$$

This low bandwidth secure channel can now be used to transmit a short secret key, which can further be used in transformation (5.15) to authenticate Diffie-Hellman half-keys.

$$A \xrightarrow{t_1} B \left. \right\} \rightsquigarrow A \xrightarrow{\text{short}} B \quad (5.17)$$

5.10 Authentication using a Shared Secret Key - User-Assisted (P7)

Devices A and B in protocol class P7 have exclusive access to a random oracle O (e.g. the user) providing random data to establish a short term secret key between A and B . This is depicted by the following transformation:

$$O \xrightarrow{t} A/B \} \rightsquigarrow A \xrightarrow{t_{short}} B \quad (5.18)$$

Note that this transformation is quite similar to transformation (5.5), where we use a random oracle to generate a long term key. The oracle here, can only send few random data, such that we only get a short shared secret key between A and B . This key is then used to authenticate Diffie-Hellman half-keys.

By combining transformation (5.15) and (5.18) we get:

$$\left. \begin{array}{ccc} O & \xrightarrow{t_1} & A/B \\ A & \xleftarrow{t_2} & B \\ & t_2 > t_1 & \end{array} \right\} \rightsquigarrow A \xrightarrow{t_2} B \quad (5.19)$$

5.10.1 Shake Well Before Use (type 2)

In Shake Well Before Use Type 2 (3.17.2), the user generates random accelerator data on A and B by shaking A and B , equipped with accelerators, in one hand. We assume that an adversary is not able to derive or influence the random data. This leads to the secure channel depicted in transformation (5.18). Both devices extract a feature vector from the random accelerator data. This can be seen, if the vectors are equivalent, as a shared secret key $A \xrightarrow{t_{short}} B$, but this key would be neither of defined length (e.g. 128 bit) nor distributed uniformly. Nevertheless the feature vectors can be used to authenticate Diffie-Hellman public keys. Shake well Before Use propose a special interlock protocol to do this. I refer to the paper [14] for a detailed description of this protocol.

5.10.2 Simple Bluetooth Pairing

In *Simple Bluetooth Pairing* the user enters a short key (PIN) on both devices. We assume this key is large enough and that the users chooses it randomly and uniformly from the set of possible keys. The user takes care that only she can enter a PIN on the device keyboards and that nobody else can see which buttons she pushes. This leads to the secure channels required in (5.18).

5.10.3 BEDA B-to-B

In BEDA B-to-B the user generates a random key on both devices by simultaneous pushing a button on both devices. We assume that only the user can push a button on a device and that it is infeasible for an adversary to determine the time between two button events. This provides the necessary channel $O \bullet \longrightarrow \bullet A/B$ for transformation (5.18).

Conclusions

We have seen that most device pairing protocols make use of public key techniques to resist against passive attackers. To defend an active attacker, the messages are sent over user-assisted channels or OOB channels with special physical properties to ensure that no adversary can affect the communication without being noticed by the user or the protocol. Other protocols directly generate a shared secret key on two devices by receiving random data provided or generated by the user or by exchanging key material over secure OOB channels.

All presented protocols need the user to check certain conditions or accomplish certain tasks. Although the integration of the user into the protocols can be very different, the functional principles of many protocols are often quite similar. This allows to divide the protocols into different classes. These classes can further be described by transformations of security primitives. This abstraction allows to better understand current protocols and to identify the necessary requirements for further ones.

Bibliography

- [1] Dirk Balfanz, D. K. Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. 2002.
- [2] Stefan Brands and David Chaum. Distance-bounding protocols. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 344–359, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.
- [3] Ileana Buhan, Jeroen Doumen, Pieter Hartel, and Raymond Veldhuis. Feeling is believing: a location limited channel based on grip pattern biometrics and cryptanalysis, 2006.
- [4] M. Cagalj, S. Capkun, and J. Hubaux. Key agreement in peer-to-peer wireless networks. *Proceedings of the IEEE (Special Issue on Cryptography and Security)*, 2006.
- [5] Mario Cagalj, Jean-Pierre Hubaux, Srdjan Capkun, Ramkumar Rengaswamy, Ilias Tsigkogiannis, and Mani Srivastava. Integrity (i) codes: Message integrity protection and authentication over insecure channels. *sp*, 0:280–294, 2006.
- [6] Claude Castelluccia and Pars Mutaf. Shake them up!: a movement-based pairing protocol for cpu-constrained devices. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 51–64, New York, NY, USA, 2005. ACM.
- [7] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [8] D. Dolev and A. C. Yao. On the security of public key protocols. In *SFCS '81: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society.
- [9] Christian Gehrman, Chris J. Mitchell, and Kaisa Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7(1):29–37, Spring 2004.
- [10] Michael Goodrich, Michael Sirivianos, John Solis, Gene Tsudik, and Ersin Uzun. Loud and clear: Human-verifiable authentication based on audio. In *IEEE ICDCS*, 2006.
- [11] Bluetooth Special Interest Group. Simple pairing whitepaper, 2006.
- [12] Yang Liu, Kaikun Dong, Lan Dong, and Bin Li. Research of the arp spoofing principle and a defensive algorithm. *WTOC*, 7(5):516–520, 2008.
- [13] Ueli M. Maurer and Pierre E. Schmid. A calculus for secure channel establishment in open networks. In *Proc. 1994 European Symposium on Research in Computer Security*

- (*ESORICS' 94*), D. Gollmann (Ed.), *Lecture Notes in Computer Science*, pages 175–192. Springer-Verlag, 1994.
- [14] Rene Mayrhofer and Hans Gellersen. Shake well before use: Authentication based on accelerometer data. In *In Pervasive*, pages 144–161. Springer, 2007.
- [15] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 110–124, Washington, DC, USA, 2005. IEEE Computer Society.
- [16] Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syver-son. Distance bounding protocols: authentication logic analysis and collusion attacks. In R. Poovendran, C. Wang, and S. Roy, editors, *Secure Localization and Time Synchronization in Wireless Ad Hoc and Sensor Networks*. Springer Verlag, 2006.
- [17] Jason Reid, Juan, Tee Tang, and Bouchra Senadji. Detecting relay attacks with timing-based protocols. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 204–213, New York, NY, USA, 2007. ACM Press.
- [18] Nitesh Saxena, Jan-Erik Ekberg, Kari Kostianen, and N. Asokan. Secure device pairing based on a visual channel (short paper). In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 306–313, Washington, DC, USA, 2006. IEEE Computer Society.
- [19] Yaniv Shaked and Avishai Wool. Cracking the bluetooth pin. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 39–50, New York, NY, USA, 2005. ACM.
- [20] Claudio Soriente, Gene Tsudik, and Ersin Uzun. Beda: Button-enabled device pairing.
- [21] Claudio Soriente, Gene Tsudik, and Ersin Uzun. Hapadep: Human-assisted pure audio device pairing. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC*, volume 5222 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2008.
- [22] F. Stajano and R. Anderson. The resurrecting duckling: security issues for ubiquitous computing. *Computer*, 35(4):22–26, Apr 2002.
- [23] Jani Suomalainen, Jukka Valkonen, and N. Asokan. Security Associations in Personal Networks: A Comparative Analysis. In Frank Stajano, Catherine Meadows, Srdjan Capkun, and Tyler Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks 4th European Workshop, ESAS 2007, Cambridge, UK, July 2-3, 2007*, number 4572 in *Lecture Notes in Computer Science*, pages 43–57, 2007.
- [24] Srdjan Čapkun and Mario Čagalj. Integrity regions: authentication through presence in wireless networks. In *WiSe '06: Proceedings of the 5th ACM workshop on Wireless security*, pages 1–10, New York, NY, USA, 2006. ACM.
-