Implementation and evaluation of a secure device pairing protocol

Master Thesis

Author(s):
Huser, Lukas

Publication date:
2009

Permanent link:
https://doi.org/10.3929/ethz-a-005797208

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master Thesis

Implementation and Evaluation of a Secure Device Pairing Protocol

Lukas Huser

March 2009

Supervised by
Iulia Ion
Prof. Srdjan Capkun

Abstract

In a world full of mobile and smart devices, enabled to communicate with each other over wireless channels, secure pairing of two devices is of prime importance. The main threat is the so-called Man-in-the-Middle (MitM) attack, where an attacker inserts itself into the pairing protocol and impersonates one of the legitimate parties. Different methods have been proposed, which do not rely on a common security infrastructure, but exploit auxiliary channels instead, and typically involve the user in the pairing process. The most common and minimal interface available on a wide variety of devices is a single button. BEDA (Button-Enabled Device Association), a protocol suite for secure pairing of devices with minimal user interfaces, can accommodate pairing scenarios where one (or even both) devices only have a single button as their "user interface". This thesis provides an implementation of different button based auxiliary channels, as well as demo applications for mobile phones (J2ME) and desktop computers (J2SE). Additionally, these channels are evaluated with respect to ease-of-use in a comparative user study.

Contents

1 Introduction

Establishing a secure communication channel between two wireless devices is a challenging problem, especially if the two devices do not share any information a priori. This is often the case for spontaneous communication between mobile and independent devices. Today, we already encounter a large variety of mobile devices in our everyday life, most prominently mobile phones and PDAs. As more and more smart devices and electronic gadgets with wireless communication abilities get available, spontaneous interactions between such devices will become much more frequent.

Those devices typically do not share a common security environment, like a globally trusted public key infrastructure (PKI), and it is questionable whether such an infrastructure will be available in the future. *Secure device pairing* denotes the process of establishing a shared secret key between two devices, previously unknown to each other. Depending on the specific use case and its security requirements, this key can then be used to build an authentic or secure communication channel through standard symmetric cryptography.

1.1 Motivating Examples

Some motivating example use cases for secure device pairing may include:

- a laptop with peripheral devices like wireless mice or keyboards
- a mobile phone with a wireless headset
- a mobile device with a public WLAN router
- a digital camera or mobile phone with a public printer.

As well as some more exotic scenarios like

- a wrist watch with an mp3 player
- a key fob (electronic key) with a car or door
- a mobile device with a digital picture frame

- wireless game pads with a video game console.

1.2 Problem Description

Classic device pairing usually requires the user to enter a (short) PIN or password to both devices. Several problems arise with this method: As users tend to choose weak passwords (e.g. too short or simply guessable), it does not guarantee a high level of security. On the other hand, it is cumbersome for a user to memorize long passwords and therefore usability will suffer.

An alternative approach to device pairing is based on the Diffie-Hellman (DH) key agreement protocol [9, p. 515–520] with manually authenticated DH public keys. The term *manual* refers to the role of the user, who usually is actively involved in the pairing process. This may include entering the same data to both devices, copying data from one device to the other or comparing data output from both devices [2]. The main threat to the DH key agreement protocol over an insecure channel is the so-called Man-in-the-Middle (MitM) attack where a malicious third party "sits in the middle", intercepts all sent messages and inserts its own. By doing so, the attacker is able to masquerade as both legitimate parties and establish its own shared key with either of them. See figure 1.1 where the malicious third party M impersonates both legitimate parties A and B and establishes two secret keys K_A and K_B with them while A and B believe to have computed a secret shared key K with each other. To avoid the MitM attack, the DH public keys must be sent over authentic channels, or, equivalently, authenticated after the key agreement in an additional step. This additional authentication step does not require to send the full length public keys over an authenticated channel, instead it usually is sufficient to exchange shorter hash values thereof. For transmitting short authenticated messages, so-called *auxiliary* or *out-of-band channels* can be exploited. Depending on the specific input and output facilities of a given device and the security assumptions on the used auxiliary channels, different authentication protocols may be applied.

1.3 Contributions

Some of the use cases mentioned above involve devices with very restricted user interfaces (as e.g. a wireless headset). Button-based auxiliary channels, as described in "BEDA: Button-Enabled Device Association" [14], are able to transfer data by pressing a single button and are therefore suited to securely pair such limited devices. The main contributions of this thesis are:

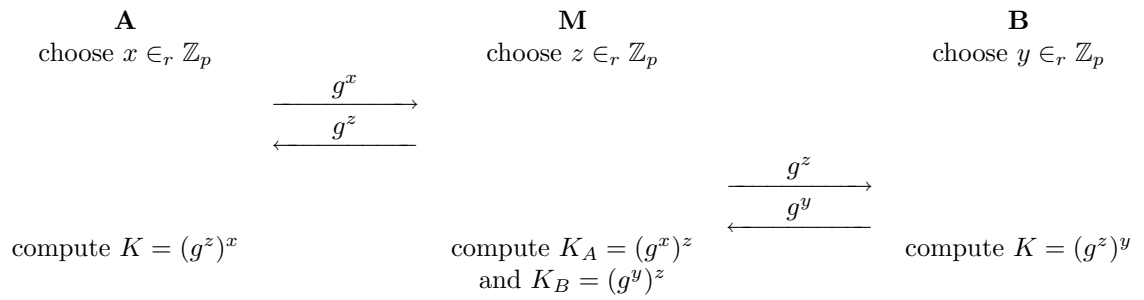


Figure 1.1: MitM attack on the DH key agreement protocol over an insecure channel

- a real world implementation of the button channels mentioned in the BEDA paper,
- a proposal and implementation of three new button-based auxiliary channels for enhanced usability,
- a more efficient (and therefore more user friendly) authentication method for the input channel which relies on synchronous button presses on both devices
- and a comparative user study which has been conducted to evaluate the implemented button channels with respect to ease-of-use.

1.4 Thesis Structure

Chapter 2 describes different out-of-band channels including the previously mentioned button-based auxiliary channels in more detail and introduces the Unified Auxiliary Channel Authentication Protocol (UACAP).

Chapter 3 proposes three new button-based auxiliary channels and explains in detail how arbitrary data can be transmitted between two devices through simple button presses and how two devices can extract the same data from a series of synchronous button presses independently of each other.

Chapter 4 shows how the different button channels have been implemented and what design decisions have been taken. This chapter provides screenshots of the implemented channels and of the demo applications.

Chapter 5 describes the user study which has been conducted to compare the usability of the implemented button channels and discusses the results and findings.

Chapter 6 finally provides a summary and draws the conclusions of this thesis.

2 Related Work

2.1 Auxiliary Channels

Depending on the hardware components of the used devices, many different out-of-band channels have been proposed. Possibilities that arise if both devices provide a display and a keypad include entering the same PIN to both devices, reading a PIN from one device and typing it to the other, and comparing two PINs which are displayed on both devices [16]. If the two devices only provide a display, a user can compare two hash values visualized as a random art pattern [12] or as non-sensical English sentences [3]. Modern mobile phones typically are equipped with speakers and a microphone as well as a camera, which can be exploited to transmit data between the two devices. One device could play an audio tune while the other is recording it, or the user could compare two melodies which are played by both devices in turn [15]. A device equipped with a camera can take a picture of a 2D bar code which is displayed on another device [8]. Two devices which have built-in accelerometer sensors can derive the same data from a common movement pattern [6].

From a user point of view, all auxiliary channels can be classified into three categories:

Input channels require the user to input the same data to both devices, for example by manually entering the same PIN to both devices or by shaking two devices together.

Transfer channels have a sending and a receiving device. The data may be transmitted directly from one device to the other (e.g. by playing an audio sequence on one device and recording it on the other) or by involving the user himself (e.g. reading a data string from one device and entering it to the other).

Verification channels make use of a common output facility on the two devices and let the user compare the emitted data. Examples include comparing two visual random art patterns or comparing two melodies played by the devices.

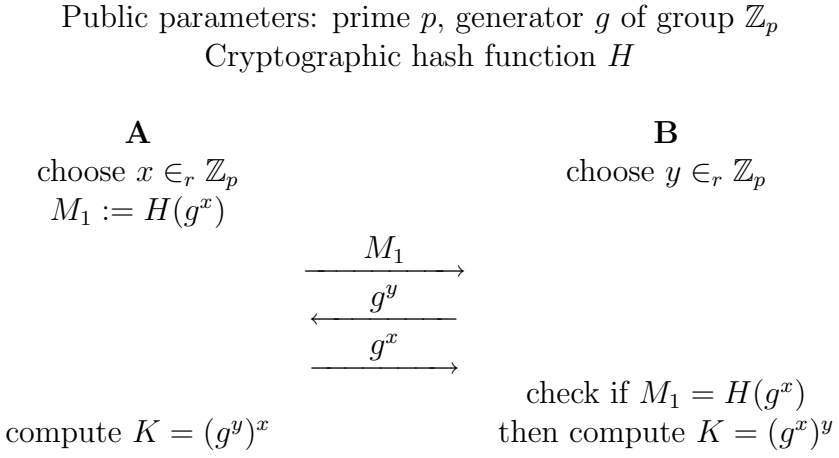


Figure 2.1: MA-DH key agreement protocol

2.2 Unified Auxiliary Channel Authentication Protocol

The Unified Auxiliary Channel Authentication Protocol (UACAP) [7] aims to provide a protocol specification which includes all different types of out-of-band channels. Depending on the type and the security properties of an auxiliary channel, the UACAP supports the following authentication modes: *long authentic transfer*, *short authentic transfer*, *short confidential input*, *short non-confidential input* and explicit user *verification*.

A protocol run consists of two main phases: key agreement over an insecure channel and subsequent verification based on an auxiliary channel. The key agreement step is the same for all different UACAP modes. It uses the MA-DH protocol [4] (see figure 2.1) which is a variant of the standard Diffie-Hellman key agreement protocol with an additional commitment step to prevent the most basic Man-in-the-Middle attacks. The key agreement phase results in a shared key K between the two devices. This key becomes a valid *shared secret key* through mutual authentication of the exchanged DH public keys (g^x and g^y) in the second phase. This thesis makes use of the *short authentic transfer* and the *short confidential input* modes of the UACAP. The term *short* refers to the length l of the exchanged out-of-band messages which can be short (about 20 bits). The UACAP ensures that any MitM attack must be performed *online* and reduces the attacker to a single one-off chance of guessing the l-bit out-of-band message, which directly results in an attack probability of 2^{-l}.

In the *short authentic transfer* mode, one device sends the hash value of the concatenated DH public keys over an authentic auxiliary channel to the other device. The receiving device computes this hash value as well, compares it to the received out-of-band message

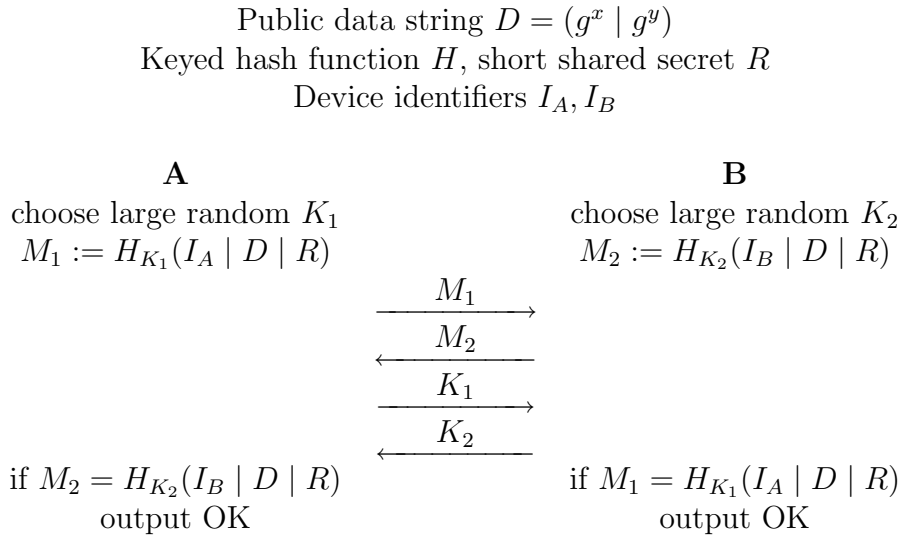


Figure 2.2: MANAIII authentication protocol

and informs the user whether the two values match. The user must then report the outcome of this comparison to the initially sending device, which corresponds to another 1-bit out-of-band message, sent in the opposite direction. For the *short confidential input* mode, the UACAP relies on the MANAIII protocol [2] (see figure 2.2). It authenticates the DH public keys based on a short shared secret value R which can be entered in advance to both devices through a *confidential* input channel. Both devices choose large random keys K_1 and K_2 and send a keyed hash of the concatenation of the secret value and the DH public keys to the other device. After receiving these messages, they will exchange the previously chosen keys as well. Both devices now can compute the same keyed hash which was sent by the other device and compare the two values. Both devices must report the outcome of this comparison to the user [17].

2.3 BEDA: Button-Enabled Device Association

In the paper "BEDA: Button-Enabled Device Association" [14], the authors describe a set of out-of-band channels which are suitable for devices with a minimal user interface, namely a single button. However, it is necessary that one of the devices provides additional interfaces (e.g. a simple display) to let the user initiate and control the pairing process and both devices should be able to signal a successful or failed pairing attempt to the user (e.g. through a green or red LED).

There are two different types of channels that can be realized: *input* and *transfer*. In

the input case, both devices require only a single button and the user will enter the same data to both devices. More possibilities arise if one of the devices has additional capabilities like vibration functionality or a larger display. In this case, one of the devices can "send" signals and the other will receive them (through button presses). This type of channel is referred to as a *transfer* channel. The amount of data that can be transmitted over a button channel directly depends on the number of button presses. Since a large number of button presses reduces the usability of this method, the capacity of these button channels is quite limited, typically about 20 bits.

Virtually the only thing one can do with a button is to press it (and release it again, for that matter). A general strategy to capture arbitrary data (e.g. as a bit string) through a single button is to record the timestamps of multiple button events (press or release) and compute the relative time differences between subsequent events. This results in a sequence of time intervals which then can be processed further. Typically, the intervals will be rounded to a given time unit and a few bits are then extracted from each interval and concatenated to the final bit string. All button-based auxiliary channels discussed here rely on this principle.

The following button channels are proposed in the BEDA paper:

Button-To-Button is a pure button channel. Both devices only require to have a single button which will be pressed by the user synchronously on both devices. By listening to those synchronous button presses (actually the intervals between subsequent button presses) independently on both devices, they are able to derive the same data from the recorded sequence of intervals. As the user is the source of the captured data and both devices take the role of a receiver, this channel is of the *input* type.

Display-To-Button requires one device to have a display. This device takes the role of the sender: It emits visual signals (by painting the screen black for a short time) to which the user reacts by pressing the button on the other device (the receiver). Like this, arbitrary data can be transmitted from one device to the other. This channel is therefore referred to as a *transfer* channel.

ShortVibration-To-Button requires one of the devices to have vibration functionality. It is similar to the Display-To-Button channel, but instead of emitting visual signals, the sending device will give a signal by vibrating for a short time. The user reacts to the vibration signals by pressing the button on the receiving device as before. Like the Display-To-Button channel, this is a *transfer* channel.

LongVibration-To-Button is similar to the short vibration channel, but instead of emitting short impulses, the sending device will alternatively vibrate and then pause again for longer time intervals. The user will press the button on the receiving device *while* the other one vibrates. The receiving device records both button presses and releases. Again, as there is a sending and a receiving device, this channel belongs to the *transfer* channels.

3 Design

In comparison to the original BEDA paper, this thesis proposes three new button-based auxiliary channels and provides improvements to the existing ones. The improvements are focused on the usability of the channels by emphazising the concepts of *feedback* and *visibility* as described by Donald Norman in his book "The design of everyday things" [10]. The three new channels aim to improve the existing Display-To-Button channel by providing more intuitive visual signals. In addition, this thesis improves the Input channel (Button to Button) by providing an efficient way to handle the independently captured data by introducing so-called "candidate secrets".

3.1 Additional Channels for Enhanced Usability

It is important to give feedback to the user (e.g. on the display) to show the effect of an action. All transfer channels (including the vibration channels) will rely on visual feedback to enhance their usability. When invoking a transfer channel, both devices briefly display an initial screen with an icon which depicts the current role: The receiving device displays the icon of a button while the transmitting device shows the icon of a mobile phone which is sending some data. During transmission, the sending device displays a counter of the already sent signals, while the receiving device displays a counter of the already processed button events. Like this, the user always can see how many signals have been sent and how many are to come, and by comparing the counters on the two phones, he can determine whether the two devices are still synchronous or if he had missed a signal.

This thesis proposes three new button channels, all of them are of the *transfer* type and they use the display to send signals to the user. The intention is that these new channels provide better usability than the original Display-To-Button channel. The basic idea is to let the user prepare to the actual visual signal by some kind of preparatory signal, such that he does not have to react to a signal that appears all of a sudden. In contrast to the Display-To-Button channel, the new channels have slightly higher requirements regarding the display, as it should be able to show for example a colored picture in an appropriate size. Current mobile phones are typically equipped with suitable displays.

As all these channels involve the display, this thesis uses different names to distinguish them. The original Display-To-Button channel will be referred to as the "Flash Display" channel.

The following new visual button channels are proposed in this thesis:

Traffic Light is a channel that will display the picture of a traffic light on the sending device. This traffic light can be in one of three states, either the red, yellow or green light is active. Most of the time the red light will be lit, indicating to the user to rest inactive. When the light turns yellow, the user can prepare and as soon as it turns green, he reacts by pressing the button on the receiving device. This channel is therefore similar to the Flash Display channel, but the simple black screen is replaced by a well known symbol from everyday life with an intuitive color scheme.

Progress Bar is a channel that follows a different concept. The sending device displays a horizontal bar, divided into alternatively colored sections (for example gray and green) which represent the different time intervals. A black progress bar grows over the sections from left to right. The user will press and hold the button on the receiving device while the black bar grows over a green section and release it on gray sections. The user can see the whole interval pattern from the beginning and he can prepare for every button event (press or release) in advance.

Power Bar is very similar to the Progress Bar channel. Instead of a single horizontal bar, it displays multiple vertical "power bars", consisting of a gray and green section each. A black progress bar grows from bottom to top over each such bar. Again, the user will press and hold the button on the receiving device while the black bar runs over a green interval and release it on gray intervals. This channel is better suited for displays which are higher than wide, while the Progress Bar channel is preferred on wide screen displays which are wider than high.

3.2 Transmitting Data

All button channels are defined by three parameters: The capacity C (the message length of a single invocation of a button channel), the smallest considered time unit t_{min} and the number of bits b which are encoded by a single time interval. The three values have implications on the security of the pairing process and the usability of the channels. It is therefore crucial to find a good balance between ease-of-use and security by carefully adjusting these parameters.

The capacity is fixed to $C = 24$ bits for all button channels in this thesis. As it corresponds to the length of the sent out-of-band message during the UACAP run, this

directly leads to an attack probability of 2^{-24}. The smallest time unit t_{min} differs from channel to channel, but it is typically set to 1 second for a transfer channel. The parameter b is fixed for all channels and is set to $b = 3$ bits per interval. C must be a multiple of b as the out-of-band message is encoded as a sequence of $n = C/b$ intervals, which should be an integral number. In this thesis $n = 24/3 = 8$ is fixed for all button-based auxiliary channels.

In the transfer case, the sending device computes the message to transmit as the hash value of the DH public keys and truncates it to the length of C bits. The message is then split into chunks of b bits each, interpreted as binary encoded digits, increased by 1 and finally multiplied by t_{min}. This results in a list of n time intervals in milliseconds, where every interval encodes b bits of the message. The sending device now emits a total of $n + 1$ signals to the user where the idle time between the signals is determined by the previously computed intervals. As a reaction to the emitted signals, the user presses the button on the receiving device. It measures the time differences between subsequent button presses and ends up with a list of n time intervals. Each interval is then rounded to t_{min}, divided by t_{min}, and finally decreased by 1. The resulting b-bit values are subsequently concatenated to the originally sent bit string. Whether the sent and received bit strings are actually the same (i.e. the reliability of the button channel) depends on the reaction delays of the user. As the receiving device rounds each interval to the nearest multiple of t_{min}, reaction delays up to $t_{min}/2$ can be tolerated. Increasing and decreasing the b-bit values by 1 is necessary, because the b-bit representation of 0 is a valid part of the message, but an interval of 0 ms can not be signaled to the user. Every interval is a multiple of t_{min} and the smallest time interval exactly corresponds to t_{min}.

The value of b has implications on the usability of the channel. A high value increases the maximum length of an interval, computed as $2^b \cdot t_{min}$, which corresponds to the maximum time a user has to wait for a signal and should not be too high. A small value would lead to shorter waiting times, but on the other hand it increases the number of button presses needed to transmit the same amount of data. With the values of $b = 3$, $C = 24$ bits and $t_{min} = 1$ second, the longest waiting time between subsequent signals is 8 seconds and a user has to react to a total of 9 signals.

Compared to conventional pairing methods, the described setting with 9 button events roughly provides the same level of security as a PIN consisting of 7 decimal digits or a password of 4 characters (small and capital letters and digits).

3.3 Handling Independent Input

In the input case, the parameter t_{min} has a slightly different interpretation. The authentication protocol for the Input channel (Button to Button) relies on a short shared secret value. This value is entered to the devices by the user through synchronous button presses. As the devices are listening to these button events independently of each other, they will end up with a similar list of intervals (measured in milliseconds). However, they will not capture exactly the same intervals, because of the less than perfect synchrony between the two hands of an average user. The devices will perform similar steps to extract a bit string from the measured intervals like in the transfer case: Round each interval to t_{min}, divide it by t_{min} and finally reduce it modulo 2^b. Again, each interval provides b bits which are concatenated to the final bit string.

t_{min} corresponds to the time that two given intervals may differ *at maximum*. It does *not* guarantee that two intervals that have a smaller difference will always be rounded to the same value. For example, with $t_{min} = 300$ ms, if device 1 measured an interval of 449 ms and the corresponding interval on device 2 happens to be 450 ms, both devices end up with different rounded intervals (300 ms on device 1 and 600 ms on device 2), even though the user input was highly synchronous. Due to the rounding difference, the devices finally end up with different secret values, which leads to the failure of the authentication protocol. This is frustrating because the pairing process may fail even for an experienced and trained user, without the possibility to predict the outcome. This problem is independent of the used time unit and it can in particular not be solved by simply increasing the smallest considered time unit t_{min}.

To address this issue, the following approach is taken in this thesis: Instead of extracting a single secret value from a given sequence of intervals, multiple "candidate secrets" are derived from the same measured data. The different candidate values are based on different time units to which the captured intervals will be rounded. During the authentication protocol run, the two devices try all available candidate secrets in turn. The protocol run succeeds if a pair of candidate secrets, computed independently on both devices, match (i.e. both devices computed the same candidate secret). In the UACAP mode for *confidential input*, the attacker has a single one-off chance to guess the pre-shared short secret. With multiple candidate secrets, it is sufficient to guess one of them. This leads to an increased attack probability by the factor c, where c is the number of supported candidate secrets. As a high number of candidate secrets increases both the success rate, but the attack probability as well, it results in a trade-off between usability and security. In this thesis, two candidate secrets are computed from the captured interval sequence. The first candidate is received by rounding the captured intervals to $t_{min} = 400$ ms, the second is based on a smallest time unit of $t_{min} = 300$ ms.

Figure 3.1 depicts the situation for three actually performed protocol runs. It lists the

measured intervals (in milliseconds) exactly as they were captured on both devices, as well as after rounding to 400 ms and 300 ms respectively. Even though the differences between the captured intervals on the two devices are small (< 100 ms), they may end up with different interval lists after rounding. While protocol run 1 fails for $t_{min} = 300$ ms, it succeeds when rounded to 400 ms. Protocol run 2 shows the opposite behavior, it fails for 400 ms and succeeds for 300 ms. With the approach of two different candidate secrets based on 400 ms and 300 ms, both protocol runs succeed as it suffices that one of the candidate secrets matches on both devices. However, there are still cases where both candidate secrets differ on the two devices (as in protocol run 3) and the authentication protocol fails.

For security reasons it is crucial that the captured data and the derived candidate secrets are random. The randomness depends on the average time that a user waits between subsequent button presses and the values of b and t_{min}. The authors of the BEDA paper noted that a user will wait about 3 to 4 seconds between button presses on average. As this thesis uses 400 ms as the highest value for t_{min} and $2^b \cdot t_{min} = 8 \cdot 400$ ms $= 3200$ ms, the input can be expected to be random.

3.4 Security Assumptions

In the BEDA paper, all channels are assumed to be *secure*, meaning both *authentic* and *confidential*. The reason is that the used authentication protocol (a variant of MANAIII) relies on a short secret value, shared by both devices. This assumption does not necessarily hold for all transfer channels, especially a channel displaying visual signals on the screen should not be assumed to be confidential. This thesis makes the assumption that all button-based auxiliary channels of the *transfer* type are *authentic*. This property is inherently given through the physical presence of the user. Since a message is transported by the user himself (by pressing a button), he can always assure that a message received by one device was actually sent by the other one. In this thesis the *short transfer* mode of the UACAP is used as the authentication protocol, which only relies on authentic channels.

The UACAP includes a mode for *non confidential input*, however, it is not suitable for the BEDA input channel, as it requires the user to enter the same data twice, once to each device, but at different points in time. Therefore the *confidential input* mode of UACAP is used with the explicit assumption that the input channel (Button to Button) is a *confidential* channel. This assumption can be justified, as an attacker would have to be able to exactly measure the timestamps of occurring button presses solely from the observation of the hardly visible finger movements.

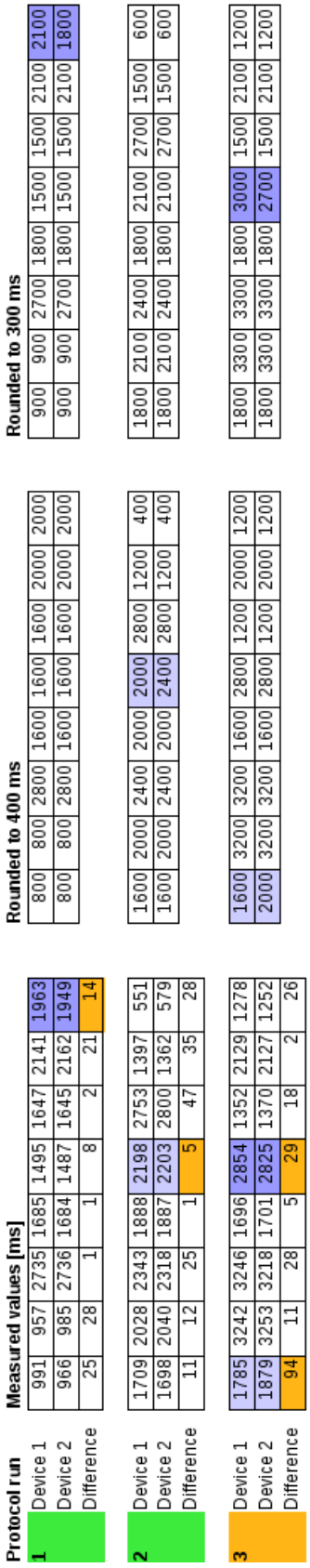


Figure 3.1: Example protocol runs for the Input channel

4 Implementation

All source code written during this thesis is available as part of the OpenUAT [5] project at `www.openuat.org` [11] or through its SVN repository respectively[1]. In particular, it includes the concrete implementations of the discussed button channels as well as two demo applications which demonstrate secure device pairing based on those channels. The two demo applications, a standard Java application and a Java MIDlet for mobile devices, are compatible with each other and allow to pair for example a mobile phone with a laptop, as long as both devices support wireless communication over Bluetooth.

All classes and their public and protected members are documented with Javadoc and important parts of the code are covered by JUnit tests. As the J2ME standard roughly corresponds to the Java language version 1.2, classes which run on both platforms refrain from using modern language constructs supported in later versions of the Java language.

4.1 Implementation Framework

The OpenUAT is an open source project licensed under the GNU Lesser General Public License (LGPL) and it aims to provide ready-to-use components for ubiquitous authentication applications. Large parts of the code are written in Java and run on J2SE as well as on J2ME. The Java toolkit provides helper classes for Bluetooth communication and service registration (`BluetoothRFCOMMServer`) as well as device discovery and service search (`BluetoothPeerManager`). For key generation and cryptographic primitives, it relies on the Java JSSE/JCE cryptographic extension or, if not available, on the Bouncy Castle Crypto API[2]. Most importantly, the OpenUAT provides an implementation of the UACAP through the `HostProtocolHandler` class.

This thesis provides an extension to the OpenUAT by adding support for button-based auxiliary channels (as described in chapters 2 and 3). To integrate the new out-of-band

[1] `https://www.gibraltar.at/svn/openuat`
[2] `www.bouncycastle.org`

channels with the existing framework, they must implement the `OOBChannel` interface which defines the two methods `transmit` and `capture`.

4.2 Button Channel Class Hierarchy

Figure 4.1 shows the UML diagram of the button channel class hierarchy. It provides an overview of the involved classes and depicts the relations between them. However, it is not a definite specification of these classes, as some methods and attributes are left out for simplicity. Some utility classes like the `IntervalList` are not shown explicitly in the diagram either. The different button channels have been added iteratively to the class hierarchy which started with the four original BEDA channels. As the channels should run on two different platforms, J2SE and J2ME, a general goal is to encapsulate all platform dependent code at a few points and to factor out the platform independent code in order to avoid code duplication.

4.2.1 ButtonChannel

The abstract class `ButtonChannel` provides a common ancestor for all concrete button channel classes. It extends the existing `OOBChannel` interface and defines important attributes and constant values which are shared by all button-based auxiliary channels. The `capture` method is basically the same for all button channels and is therefore implemented in this class as well.

To allow the abstraction of different button channels to be independent of the supported platform, the *bridge pattern* [1, p. 87–96] is applied. The `ButtonChannel` class delegates its implementation to the abstract `ButtonChannelImpl` class which provides a broad interface that allows to implement the concrete subclasses of the `ButtonChannel` class. Every supported platform will provide its own subclass of `ButtonChannelImpl`, implementing the specified interface and effectively encapsulating all platform dependent code at a single point. If a certain method cannot be implemented on a specific platform (like e.g. the `vibrate` method on J2SE), the method body should be left empty and an application should not call the respective method. This allows to define the different button channels in a platform independent manner and a new channel can be easily added by extending the `ButtonChannel` class, provided the interface of `ButtonChannelImpl` supports the needed functionality.

All button channels share two fundamental properties, the capacity and the number of bits that can be extracted from a single interval. They are defined as the public constants `MESSAGE_LENGTH` and `BITS_PER_INTERVAL` within the `ButtonChannel` class. The capac-

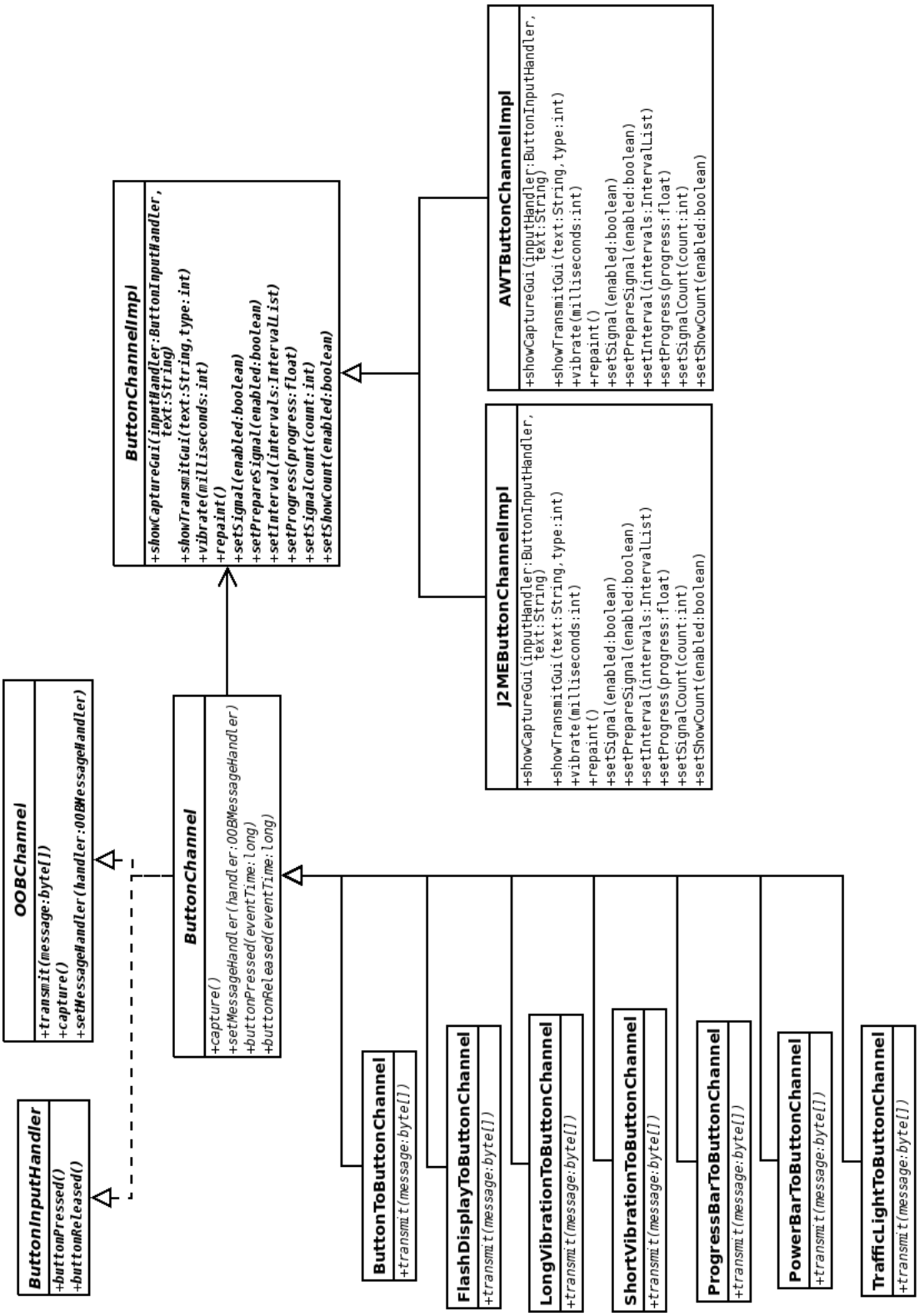


Figure 4.1: Class hierarchy of the button channels

ity is set to 24 bits (3 bytes) and from each interval 3 bits will be extracted. These two values are basically independent of each other, but the capacity should be a multiple of the number of extracted bits. The reason is that the number of time intervals needed to transmit a given message is computed as `MESSAGE_LENGTH/BITS_PER_INTERVAL`, which should be an integral number. This leads to $24/3 = 8$ intervals or, equivalently, 9 button events (presses or releases) to transmit the whole message. This class defines other important attributes, like `minTimeUnit` or `inputMode`, which differ from channel to channel and should be initialized in the concrete subclasses. The attribute `inputMode` takes one of the values `MODE_PRESS` or `MODE_PRESS_RELEASE` and defines whether a button channel only considers button presses or button releases as well.

Most importantly, this class provides the complementary methods `bytesToIntervals` and `intervalsToBytes`. The method `bytesToIntervals` is called by the sending device to convert a given `byte[]` (the out-of-band message) to a list of time intervals which then is signaled to the user. The method `intervalsToBytes` performs the reverse operation and extracts a `byte[]` from the recorded list of intervals on the receiving device. Both methods take the minimum interval size as a parameter (`minInterval`) which corresponds to the smallest considered time unit `minTimeUnit` of the specific channel. The behavior of the `intervalsToBytes` method can be a influenced through the boolean parameters `roundDown` and `useCarry`. Every interval will be rounded to a multiple of `minInterval`. If `roundDown` is set to `true`, an interval is rounded down, else it is normally rounded to the nearest multiple of `minInterval`. The parameter `useCarry` determines whether the rounding differences are propagated to the next interval. If set to `true`, it assures that rounding differences are not cumulated over multiple intervals and that the tolerated reaction delay is the same for all emitted signals. This parameter is therefore set to `true` for all transfer channels.

4.2.2 ButtonChannelImpl

The `ButtonChannelImpl` class and its subclasses provide the functionality to interact with native GUI elements on the respective platform. This includes listening to key input events and painting to the screen (e.g. on a `Canvas` object, which is available both on J2SE and J2ME with similar interfaces). The key events are not consumed directly, but delegated to a `ButtonInputHandler` instance which can react to the input events appropriately. For rendering graphical elements on the screen, e.g. giving visual signals on the sending device, every `ButtonChannelImpl` instance keeps an internal state which is represented on the screen. The methods provided to manipulate this internal state, like the `setSignal(boolean)` or `setProgress(float)` methods, do not have an immediate effect on the screen. Instead, all changes will become visible after the next call to the `repaint` method.

As mobile phones and laptops typically provide a keypad with many keys, an imple-

mentation for the respective platform must decide on a key which will represent the "single button". This decision is taken in the subclasses of the `ButtonChannelImpl` class. The implementation for J2SE listens to key events of the space bar as it is one of the most prominent keys on the keyboard and well suited to represent the single button. Every device which supports the J2ME standard provides a key which is denoted as the "fire button", due to its intended use in video games. It usually is prominently placed on the phone, centered between the navigation keys, and is typically larger than other keys. The "fire button" is therefore used as the single button for user input on J2ME.

4.3 Handling Key Events

The instances of the `ButtonChannelImpl` class are responsible for listening to key events during the capturing process. Whenever the user presses or releases a key, the operating system delivers the corresponding key event to the running application. When a key is pressed and held down, most operating systems generate and deliver additional virtual key events at a user defined key repeat rate. An implementation of a button channel should only react to user triggered key events and must ignore the artificially created events from the operating system. J2ME supports a separate event type for these virtual key events (`keyRepeated`) and an application can easily ignore them by simply listening to the `keyPressed` and `keyReleased` events. However, on J2SE the virtual key events are delivered as standard key press or key release events and different operating systems do not deliver them in a consistent way. Figure 4.2 depicts the behavior on different systems. Windows delivers a `keyPressed` event, keeps sending `keyPressed` events at the system's key repeat rate and finally delivers a `keyReleased` event. Linux delivers a `keyPressed` event followed by a pair of `keyReleased` and `keyPressed` events (in that order, but with identical internal timestamps), and finally sends a single `keyReleased` event. Most operating systems have the key repeat feature enabled by default, but a user could turn it off as well. In this case, no virtual key events are generated and a single `keyPressed` event and a single `keyReleased` event are delivered to the application. The implementation for J2SE (`AWTButtonChannelImpl`) can deal with these three different behaviors and was tested on Linux (Ubuntu 8.10) and Windows XP.

4.4 Logging Framework

In the OpenUAT, logging statements are consequently used throughout the whole project to log errors and warnings. Especially on mobile phones, the logging framework can be used as a convenient debugging mechanism as well and, most importantly, logging state-

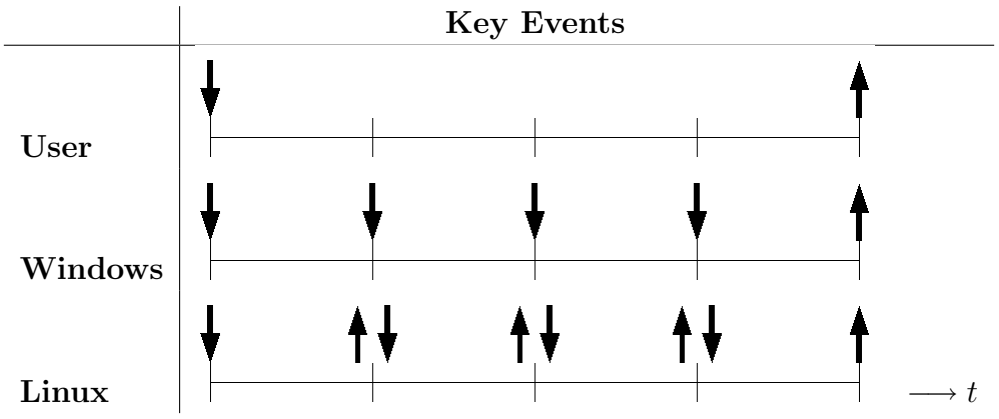


Figure 4.2: Key repeat behavior on different operating systems

ments which are written to a file can be used to collect statistical data (e.g. during a user study). The OpenUAT uses different logging frameworks for the different platforms, as there does not exist a mature logging framework which runs under both J2SE and J2ME. The reason is that virtually all logging frameworks available on J2SE rely on the Java Reflection API which is not supported on J2ME. This thesis provides a thin wrapper around the existing logging frameworks and allows any class to retrieve a suitable logger instance without referring explicitly to a platform dependent framework. This is achieved by applying the *abstract factory pattern* [1, p. 151–162]. The single product of the abstract factory represents a common abstraction of a "logger" and is defined by the `Log` interface. The concrete products are defined by thin wrapper classes around native logger instances of the supported logging frameworks. A concrete factory class is provided for every logging framework which is responsible for the creation of the respective logger instances. The `LogFactory` must be initialized at application startup through the `init` method by providing an instance of a concrete factory. After initialization, any class can retrieve a logger instance through the call to the static method `LogFactory.getLogger(String)`.

The used logging frameworks are log4j[3] on J2SE and microlog[4] on J2ME. Figure 4.3 shows the UML diagram of the factory pattern for the two mentioned logging frameworks.

4.5 Input Channel

The `ButtonToButtonChannel` class provides the implementation of the *input* channel which listens to synchronous button presses and therefore does not implement the

[3] `logging.apache.org/log4j`
[4] `microlog.sourceforge.net`

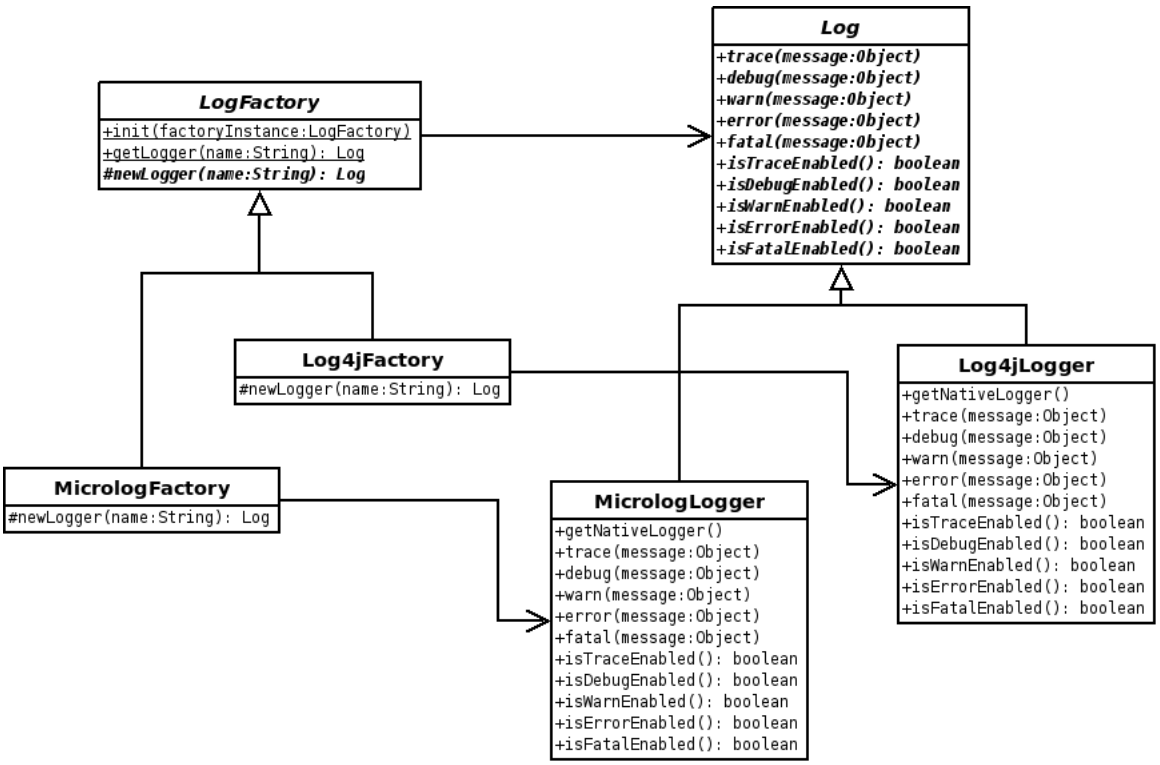


Figure 4.3: Factory pattern for the logging framework

`transmit` method. This channel is considered to be *confidential* and must not leak any information about the entered data. It does not give any feedback about the user input on the screen and it does in particular not provide a counter which displays the remaining number of button presses to the user. Figure 4.4 shows the screen which is presented to the user during the capturing process, it is the same on both devices. This channel only reacts to button presses and ignores button releases. From the captured data, two candidate secret values are computed. This is achieved by initializing the attributes `minTimeUnit` to 400 and `minTimeUnit2` to 300 milliseconds. The `capture` method, which is inherited from the `ButtonChannel` class, then takes care of computing the two candidate secrets and delivers the concatenation of the values to the `OOBInputHandler` instance.

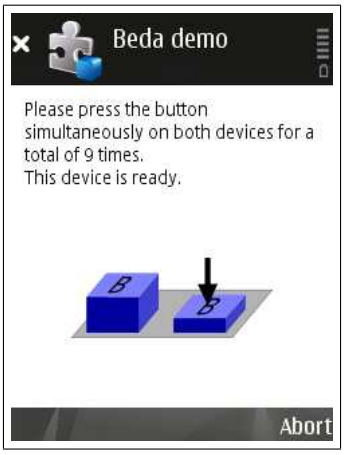


Figure 4.4: Screenshot of the Button to Button channel on J2ME

4.6 Transfer Channels

4.6.1 Flash Display

This is a *transfer* channel which displays a simple visual signal on the sending device. The signal is a black rectangle (see figure 4.5b) which is visible for 500 ms. This causes a "flashing" effect on the screen, hence the name of the channel. The receiving device only considers button presses and ignores button releases. The parameter `minTimeUnit` is set to 1000 ms which tolerates a reaction delay of the user up to 500 ms. The basic implementation of this channel does not provide any means to the user to prepare to the actual signal, the black rectangle will appear all of a sudden on the screen. A variant of this channel can be obtained by providing the boolean parameter `usePrepareSignal` to the constructor of the class. This variant displays a preparatory signal before the actual

signal on screen. The preparatory signal is again a simple rectangle, but in gray and a bit smaller than the real signal (see figure 4.5a). This channel requires the sending device to provide a display, but it may be quite limited regarding its size and the ability to display colors.

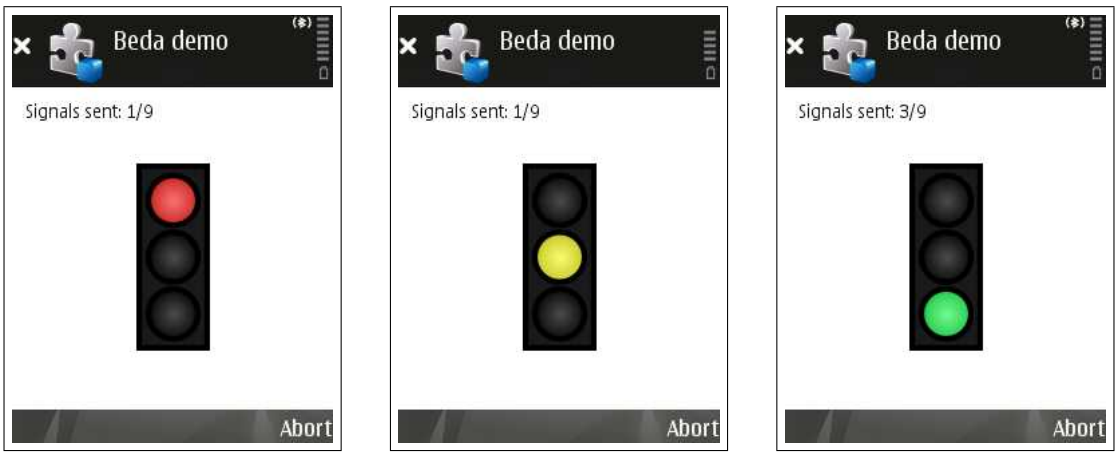
Both the sending and the receiving device display a counter on the screen which informs the user about the already sent signals and the number of processed button events respectively. On the receiving device, this information is graphically represented as a "wheel" of triangles as well (see figure 4.5c).

(a) Preparatory signal (b) Actual signal (c) Receiving device

Figure 4.5: Screenshots of the Flash Display channel on J2ME

4.6.2 Traffic Light

This channel aims to improve the usability of the Flash Display channel. It uses a traffic light as a well known symbol with an intuitive color scheme on the sending device. The traffic light may be in one of three states: red, yellow or green. If it is red, the user should wait, when it turns yellow he can prepare and as soon as it turns green he should press the button on the receiving device. Like the Flash Display channel, it considers only button presses and ignores button releases. The yellow light is shown for 350 ms and the green light is active for 500 ms. The `minTimeUnit` is set to 1000 ms to tolerate up to 500 milliseconds of reaction delay. It also correctly handles the case where the user presses the button too early (e.g. already during the yellow phase). Figure 4.6 shows the three states of the traffic light on the sending device. The screen on the receiving device looks similar to the one of the Flash Display channel (see figure 4.5c).



(a) Red: wait (b) Yellow: prepare (c) Green: react

Figure 4.6: Screenshots of the Traffic Light channel on J2ME

4.6.3 Progress Bar

The class `ProgressBarToButtonChannel` implements a different type of a *transfer* channel. The transmitting device draws a bicolored bar (gray and green) on the screen which is a graphical representation of the time intervals to be transmitted. A black progress bar constantly grows over the pattern from the left to the right (see Figure 4.7 for a screenshot). The user will press and hold the button when the black bar grows over a green interval and he releases it on gray intervals. The receiving device listens to both button press and release events. The advantage of this channel is that the user sees the whole pattern from the beginning and can prepare for every button event (press or release). The smallest time unit (`minTimeUnit`) is set to 600 ms which tolerates differences of 300 ms of either too early or too late button events. To work well, this channel requires an appropriately sized color display and is suited for wide screens. Figure 4.7 shows a screenshot on a Nokia N95 mobile phone with a horizontally flipped screen.

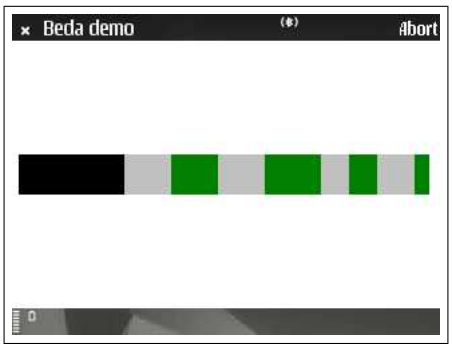
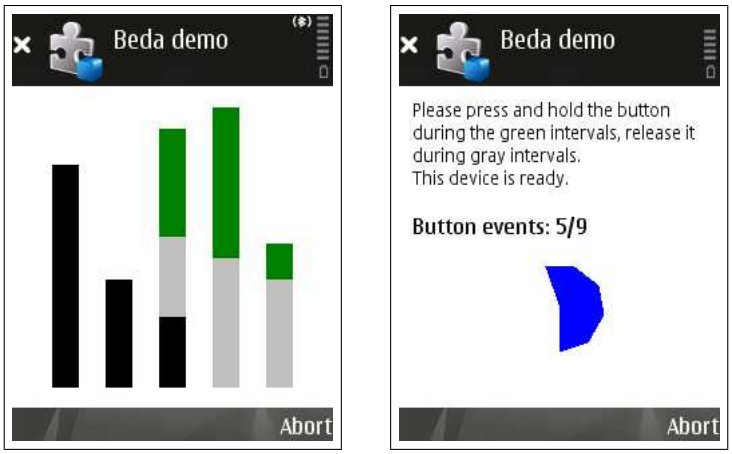


Figure 4.7: The Progress Bar channel on a horizontally flipped screen (J2ME)

4.6.4 Power Bar

This channel is similar to the Progress Bar channel, but instead of representing the time pattern as a single horizontal bar, it is split into multiple vertical "power bars". Each power bar represents two time intervals and consists of a gray and a green part which correspond to those intervals (see figure 4.8a for an example). A black progress bar grows over each vertical bar, from bottom to top, starting with the leftmost power bar. Similarly to the Progress Bar channel, the user will press and hold the button on the receiving device while the black bar grows over a green interval and he releases the button on gray intervals. The receiving device has a press-release characteristic and listens to both types of button events. The parameter `minTimeUnit` is set to 600 ms like in the Progress Bar channel.



(a) Sending device (b) Receiving device

Figure 4.8: Screenshots of the Power Bar channel on J2ME

4.6.5 Vibration Channels

All *transfer* channels described so far make use of the display to emit signals to the user. The two vibration channels instead rely on the vibration functionality of mobile devices and are implemented in the two classes `ShortVibrationToButtonChannel` and `LongVibrationToButtonChannel`. The Short Vibration channel emits short signals of 500 ms each at every interval boundary and the user reacts to them by pressing the button on the receiving device. The capturing device only considers button presses and ignores button releases. The Long Vibration channel alternates between vibrating and idle intervals. The user will press and hold the button on the receiving device while the sending device vibrates. This channel therefore has a press-release characteristic. Both vibration channels have the smallest considered time unit (`minTimeUnit`) set to 1000 ms and tolerate reaction delays of 500 ms.

4.7 Demo Applications

There are two demo applications which demonstrate the usage of the implemented button channels: A MIDlet for Java enabled mobile devices (`BedaMIDlet`) and a standard Java application for desktop computers and laptops (`BedaApp`). Both applications provide the same functionality and are fully compatible with each other. They rely on wireless communication over Bluetooth for the key agreement and exploit the button-based auxiliary channels to authenticate their respective DH public keys. On application startup, a Bluetooth server is launched which advertises the BEDA authentication service. The device is then ready to accept incoming pairing requests from a remote device. The user interface allows to search for nearby Bluetooth devices and to initiate the pairing process with devices which are running a compatible application.

The BEDA MIDlet initially displays a start screen (see figure 4.9a). The menu item "Start authentication" in the "Options" menu causes the device to scan for nearby Bluetooth devices and listing them on the screen. The user selects the device he wishes to pair with and then selects one of the available button channels (see figure 4.9b). The two devices will then run the key agreement protocol in the background and invoke the previously selected button channel to authenticate the DH public keys as specified by the UACAP. For *mutual authentication* based on a *transfer* channel, it is important that the application not only invokes the auxiliary channel to send the out-of-band message, but also informs the user about the authentication success on the receiving device and let the user report the result to the sending device. This in fact is another 1-bit out-of-band message transmitted by the user, but in the opposite direction.

(a) Start screen

(b) List of available button channels

(c) Success information

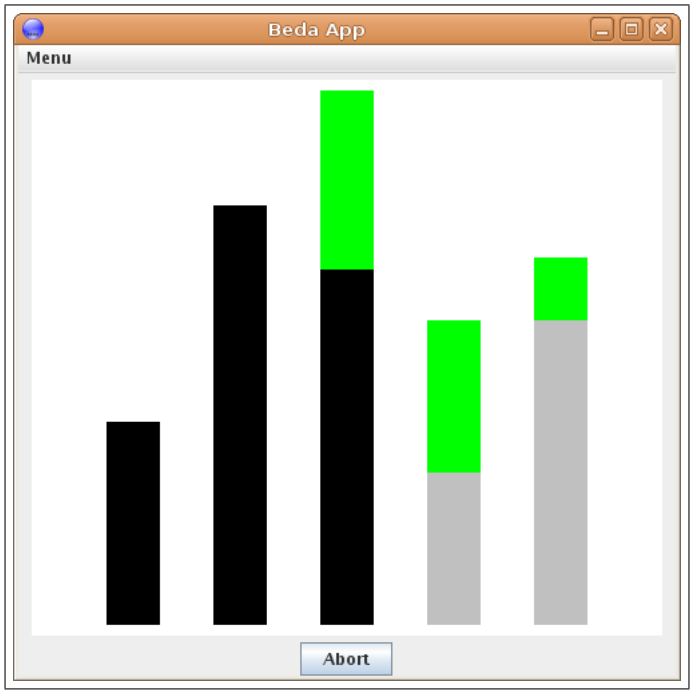
Figure 4.9: Screenshots of the BEDA MIDlet

The desktop application provides the same functionality as the MIDlet. The GUI consists of a single screen containing two lists, one for available Bluetooth devices and one

for the implemented button channels (see figure 4.10a for a screenshot). The "Refresh list" button can be used to initially populate the list of Bluetooth devices or to refresh it later on. The list of channels is initially disabled (grayed out), double-clicking a device from the list on the left enables it if the BEDA authentication service is running on the selected device. The user can now choose one of the button channels by again double-clicking on it. Figure 4.10b gives an example of how the Power Bar channel looks like on the J2SE platform.

When invoking a transfer channel, there is always a sending device which emits the out-of-band message to the user, and a receiving device which listens to button input events. In the two demo applications, the sending device is always the device which initiated the whole pairing process. This is important for those use cases where a mobile phone or a laptop is paired with a potential single-button device where it is crucial that the button is pressed on the responding device. Scenarios where this aspect might be important include the pairing of a laptop with a Wi-Fi router or a mobile phone with a Bluetooth headset.

(a) Main screen



(b) Power Bar channel

Figure 4.10: Screenshots of the BEDA Application on J2SE

5 Evaluation

In order to analyze the usability of different button-based device association methods, a comparative user study [13] was conducted. The goal was to compare the different button channels which have been implemented with respect to ease-of-use. The following channels were tested in the user study: Input (Button to Button), Flash Display, Short Vibration, Long Vibration, Traffic Light and Power Bar. The Progress Bar channels is not included in the user study because of its similarity to the Power Bar channel and to limit the number of tested channels.

5.1 Research Questions

In the user study, the following questions are addressed:

1. Is one of the general methods—input or transfer—favored over the other one (independent of the used transfer channel)?
   From a user point of view, these two methods differ fundamentally: In the input case, the user can decide himself when to press a button, he is acting instead of reacting. He can influence the total duration of the protocol directly. But the input channel suffers from the problem that it might fail, even though the user input was highly synchronous on both devices. In the BEDA paper, the input channel was rated as the hardest to use from all tested channels. Since we generate two candidate secrets out of an interval list and the protocol run fails only if both candidates fail, we expect the success rate to be higher, which may be reflected in the user feedback.

Input	Input (Button to Button)
Transfer	Flash Display, Short Vibration, Long Vibration, Traffic Light, Power Bar

2. How does ease-of-use compare between the original BEDA channels to the additionally implemented channels with improved graphical feedback on the GUI?
   We expect the Traffic Light and Power Bar channels to be more user friendly than the Flash Display channel. It is interesting to compare these two new "visual" channels with the two vibration channels as well.

Original BEDA	Flash Display, Short Vibration, Long Vibration
New	Traffic Light, Power Bar

3. How does ease-of-use compare between channels with a press-press characteristic to those with a press-release characteristic?

Press-Press	Flash Display, Short Vibration, Traffic Light
Press-Release	Long Vibration, Power Bar

4. What is the preferred channel of a user?
5. What is the average number of pairing attempts needed to successfully pair two devices for each channel?
6. What is the average overall time needed to pair two devices for each channel?

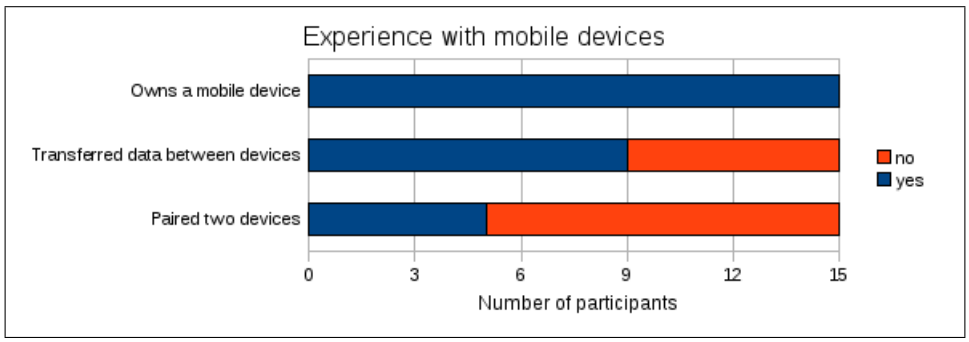
5.2 Location and Setup

The participants were recruited on the ETH campus (e.g. in the cafeteria) and from within the authors own circle of acquaintances. The study was carried out on two mobile devices (Nokia N95 mobile phones) with the BEDA demo application installed on them. This allowed to run the test sessions right where the participants have been recruited and it had the advantage that it took place in a real world environment. The demo application on the phones was available with texts in English only, but the questionnaire form of the study was in German, which was easier to understand for most participants.

A total of 15 participants have been recruited for the study. Figure 5.1 shows an overview of the participant characteristics. Most of the participants were students between 20 and 30 years old. However, two participants aged 50 years or older could be recruited as well. In figure 5.1b, the term “basic education” corresponds to the German term “obligatorische Schulbildung”. All participants owned a mobile phone and about two thirds already had transferred data between two mobile devices. About half of those people who have exchanged data between two devices had paired the devices beforehand (e.g. by typing the same PIN to both devices).

5.3 Methodology

For each of the button channels mentioned above, the following task was defined: Try to pair the two phones using the given out-of-band channel. If a pairing attempt is successful, proceed with the next task. If a pairing attempt fails, retry to pair the devices with the same channel. If the devices could not be successfully associated after



(a) Experience with mobile devices

Age	
18-24	2
25-29	10
30-34	0
35-39	1
40-49	0
50+	2

Gender	
Male	9
Female	6

Education	
Basic Education	3
Gymnasium	3
Bachelor/Master	8
Doctorate	1

(b) Participant profile

Figure 5.1: Participant characteristics of the user study

four trials, proceed with the next task. This leads to a total of six different tasks which could be performed by a participant. However, to reduce the duration of a test session, one participant only tested four out of the six available channels. Since each participant performed multiple tasks, the study followed a within-subjects design. It was therefore important to counterbalance the order in which the different tasks were performed to mitigate possible learning effects.

There are $\binom{6}{4} = 15$ different possibilities to combine 4 out of 6 channels. With 15 participants, it was guaranteed that every channel was tested equally many times, namely 10 times. To perfectly balance out the task order, we would have needed much more participants (exactly 360), which was not feasible. Therefore the order in which the tasks were performed was chosen at random. Table A.1 defines the task order for each of the 15 participants. The table is based on random permutations of the tasks (computed with MATLAB) and then manually adjusted, such that each channel appears 2 to 3 times at each position. Each row is a unique combination of four channels.

5.4 Session Outline and Timing

In total, a test session was about 20 minutes long.

Introduction 2 minutes

Each test session started with a short introduction to secure device pairing and the BEDA channels, including some examples where these button channels might be useful (e.g. pair a phone with headset, a key fob with a door or car, a mobile device with a WLAN router, a wrist watch with an mp3 player etc.). After this introduction, a participant was asked to fill in the background questionnaire (see figure A.1).

Tasks 15 minutes

The participant performed four different tasks (i.e. pairing the devices with four different button channels) in sequence. Before a candidate used a specific channel, he received a short explanation on how to handle it and what it will look like.
For the transfer channels, the moderator assured that the participant knew the following:

- which device will send the signals
- what kind of signal it is (e.g. vibration, green light, green bar etc.)
- on which device to press the button
- which button should be pressed
- whether the channel has a press-press or press-release characteristic

For the input channel, the moderator assured that the participant was aware of

- which button should be pressed
- that only button presses are considered (not button releases)
- that the input intervals should be random (not like a monotone rhythm)

Debriefing 3 minutes

After the tasks have been performed, the participant was asked to fill in the post-test questionnaire (see figure A.2). Most of the participants gave some oral feedback in a short discussion as well.

5.5 Measurements

In order to answer the research questions, the following data was collected: Qualitative information about usability and user feedback was collected through a post-test questionnaire which all participants had to fill in after they finished their tasks. Additionally, quantitative information about each pairing attempt was logged on the mobile

devices and written to a file. This data includes the duration of each pairing attempt (in milliseconds) and whether the pairing was successful or not. The log also contains the messages that were sent over the involved out-of-band channels, encoded as a list of time intervals.

For security reasons, the verification protocol for the Input channel requires the user input to be random. The logged data (especially the computed candidate secrets) allows to analyze the user input with respect to its randomness. The average interval size (the time a user waited between subsequent button presses), was 1.95 seconds with a sample standard deviation of 1.39 seconds. During 11 protocol runs with the Input channel, a total of 88 3-bit segments have been captured (one 3-bit segment for each captured interval) which build the candidate secrets. The 3-bit values for candidate secret 1 were derived by rounding the intervals to 400 ms, those for candidate secret 2 were derived by rounding the intervals to 300 ms. To test whether the observed frequencies differ significantly from the expected uniform distribution, a χ^2-test is performed in section A.4 of the appendix. With 95% statistical significance, the randomness hypothesis cannot be rejected.

5.6 Results and Findings

Figure 5.2 shows an overview of the user feedback regarding the usability of the tested channels and Table 5.1 gives a summary of the logged data. In addition, figure 5.3 shows the user preference of the tested channels. It is notable that none of the channels has been rated "very hard" by any user. All participants have been able to pair the two devices within a maximum of four trials for a given channel. The average completion times range from 47 seconds (Button to Button) to 99 seconds (Flash Display). In comparison to the user study in the original BEDA paper, the overal completion times for the transfer channels are about ten to fifteen seconds higher. This had to be expected as the channels in this thesis use a slightly higher capacity (24 instead of 21 bits). The Input channel (Button to Button) was about six seconds faster than in the original study. The denoted completion times do not only include the time for out-of-band transfer, but also include the time needed for wireless communication over Bluetooth (about 1.5 to 2.5 seconds per protocol run) and for cryptographic processing (about 2 to 3 seconds per protocol run).

A general conclusion from the discussions after the test sessions is that most users either prefer the vibration channels over the visual channels or vice versa. Some people find it hard to concentrate on the screen and react to visual signal while they find it convenient to react to haptic signals. In contrast, other participants disliked the vibrations and were much more comfortable with the visual signals.

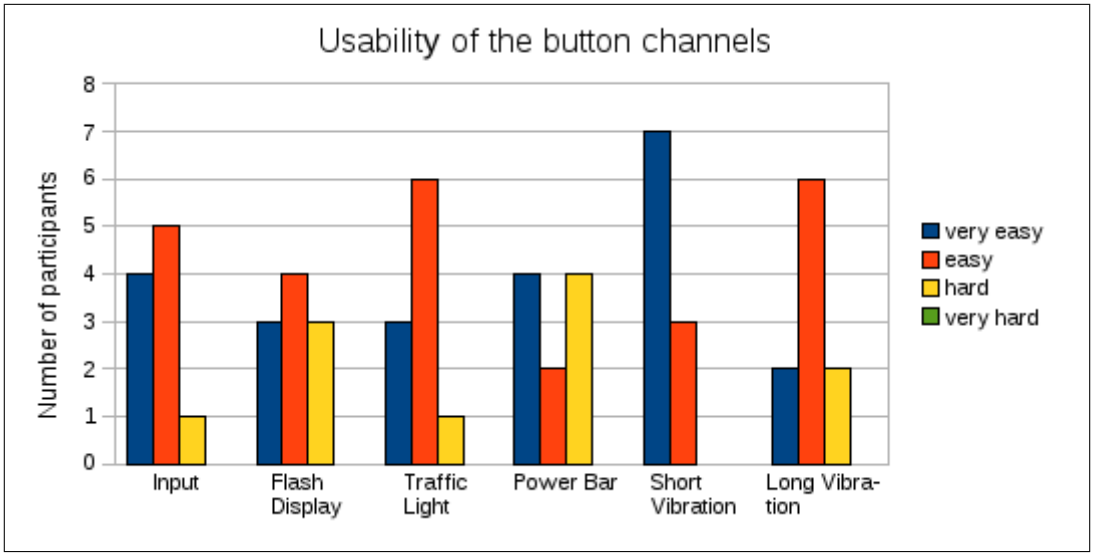


Figure 5.2: Usability of the tested button channels

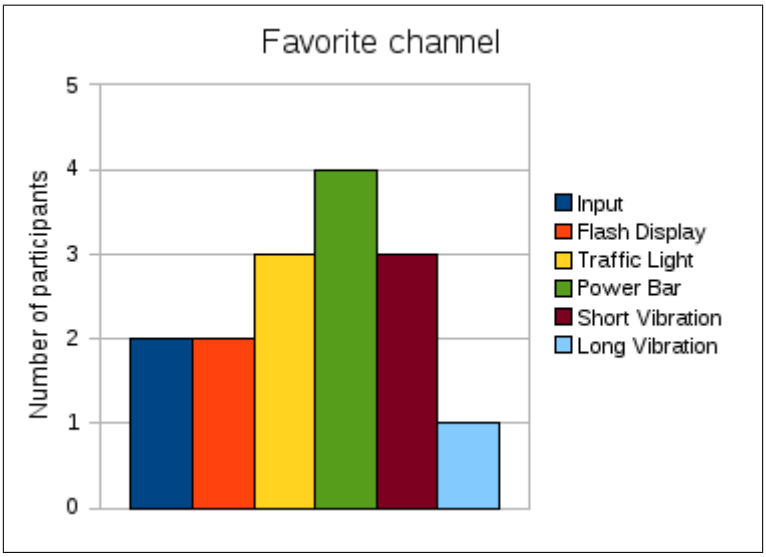


Figure 5.3: User preference

Method	Average completion time in seconds	Average number of retrials
Input	47.27 ($sd = 27.23$)	1.1 ($sd = 0.32$)
Flash Display	98.98 ($sd = 52.04$)	1.5 ($sd = 0.71$)
Traffic Light	69.66 ($sd = 31.38$)	1.1 ($sd = 0.32$)
Power Bar	86.27 ($sd = 50.51$)	1.9 ($sd = 1.1$)
Short Vibration	63.76 ($sd = 7.42$)	1 ($sd = 0$)
Long Vibration	65.62 ($sd = 27.6$)	1.2 ($sd = 0.42$)

sd = sample standard deviation

Table 5.1: Summary of the logged data

Vibration channels The Short Vibration channel performed best from all tested channels. It has the best usability rating (all participants found it "very easy" or "easy" to use this channel) and has a high user preference as well. Furthermore, all participants were able to pair the two devices in the very first pairing attempt when using the Short Vibration channel. In contrast, the Long Vibration channel did not perform that well. The general usability was rated lower (still, many people found it "easy" to use) and the user preference is notably weak for this channel. During discussion, some people mentioned that they disliked the long vibration durations of up to eight seconds. This difference is interesting because these two channels were rated exactly the same in the original user study in the BEDA paper.

Visual channels When comparing the "visual" channels, the Traffic Light channel performed better than the Flash Display channel. It has better overall usability, a higher success rate and a higher user preference as well. The Power Bar channel did not perform that well and it has in particular the highest failure rate of all tested channels (1.9 retrials until success). In comparison to the other transfer channels, the Power Bar channel used a smaller time unit of 600 ms instead of 1000 ms. This choice was based on the fact that the Power Bar channel has a different mode of operation than the other transfer channels and on the positive experience with such a smaller time unit made by the author of this thesis. However, it turned out that this small time unit did not work well for all participants. About half of the participants were able to pair the two devices in the first attempt with the Power Bar channel, others needed three or even four trials. This leads to the high failure rate and average completion time shown in Table 5.1 and is reflected in the usability feedback, where participants found it either "very easy" or then "hard" to use this channel. Despite the relatively poor performance, the Power Bar channel has the highest user preference of all channels. This leads to the hypothesis that the Power Bar channel, with a higher "smallest considered time unit" (e.g. 1000 ms), could perform equally well or even better than the Traffic Light channel.

Protocol run	1	2	3	4	5	6	7	8	9	10	11
Candidate secret 1 matched	yes	yes	yes	no	yes	no	no	no	no	yes	yes
Candidate secret 2 matched	yes	yes	yes	yes	yes	yes	yes	yes	no	no	no
Authentication successful	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes

Table 5.2: Candidate secrets for the Button to Button channel. Candidate secret 1 is based on 400 ms, candidate secret 2 on 300 ms.

The high user preference towards the Traffic Light and Power Bar channels coincide with the feedback given during the short discussion after a test session. Two users explicitly mentioned the nice and intuitive color scheme of the Traffic Light channel, others appreciated the preparation time or liked the traffic light as a common, well known symbol from everyday life. One participant criticized the duration of the yellow light as too short. For the Power Bar channel, participants positively mentioned the fact that the total timing pattern is visible from beginning and that there is no need for sudden reactions. A participant chose this channel as his favorite because it was the most "interactive" channel, another mentioned the coloring scheme with gray and green as an intuitive and appealing combination. One participant chose the Flash Display channel as his favorite channel on the grounds that it is "easy to use but challenging as well".

Input channel The Input channel (Button to Button) performed quite well and has a similar rating like the Traffic Light channel, even though it seems not to be most user's preferred channel. The performance of this channel is in particular notable when comparing it to the initial user study in the BEDA paper. The average number of retrials for success has been reduced from 2.45 in the BEDA paper to 1.1 in this thesis. With the higher success rate, the overall usability has increased as well and, in contrast to the BEDA paper, the Button to Button channel was rated more user friendly than the Flash Display and Long Vibration channels. The higher success rate is a result of the new concept of the so-called "candidate secrets". Table 5.2 shows whether the two devices computed the same candidate secrets for the eleven protocol runs of the Button to Button channel during the user study. Only one protocol run failed (nr. 9), where both devices derived different values for both candidate secrets. When looking more closely at the captured data for this particular protocol run, it is interesting to note that it was not the "user's fault" that the authentication protocol failed. The user input was highly synchronous (the maximum difference between two intervals on the two devices was about 30 ms), the intervals just happened to fall on critical rounding boundaries for both considered time units.

The current study was conducted on two mobile phones of the same type (Nokia N95), while the study in the BEDA paper was performed on two different mobile phones. Presumably it is easier to press the button on two devices simultaneously if they are

both of the same type. It feels the same for both hands and the two buttons have the same characteristics and need the same amount of pressure to be applied. However, it is not clear how much (if at all), the good performance of the Button to Button channel was influenced by this fact.

Transfer versus Input The Input channel (Button to Button) does not generally fall behind the transfer channels when comparing their overall usability, but the user's preferences tend towards transfer channels like Short Vibration, Power Bar or Traffic Light. It is certainly a valid option for use cases where none of the involved devices provides a decent color display or vibration functionality. Two participants were irritated about the missing visual feedback about the number of button presses when using the Input channel. On the other hand, one participant found it generally hard to react to any signal from the sending device and therefore preferred the Input channel where he could choose himself when to press the button.

Press-Press versus Press-Release There seems to be no clear conclusion on this point. While the Long Vibration channel performed worse than the Short Vibration channel, the Power Bar channel could perform much better, probably equivalent to or better than the Traffic Light channel. As none of the participants explicitly mentioned this characteristic (i.e. whether only button presses were counted or button releases as well) in the discussion, it seems not to be a crucial point which influences the usability of a channel.

6 Conclusions

This thesis provides an implementation of button-based auxiliary channels, suited for spontaneous, secure pairing of devices with minimal interface capabilities. In the simplest case, both devices only require to provide a single button. The button channels are implemented as an extension to the OpenUAT project [11] and aim to provide reusable components for ubiquitous authentication applications.

6.1 Discussion and Limitations

The user study showed that users prefer the three transfer channels Short Vibration, Traffic Light and Power Bar. While the Short Vibration channel was already introduced in the paper "BEDA: Button-Enabled Device Association" [14], the Traffic Light and Power Bar channels have been developed during this thesis. The Input channel (Button to Button) provides an interesting alternative for scenarios where none of the devices provides an appropriate color display or vibration functionality. The concept of "candidate secrets" introduced in this thesis provides a higher reliability of the Input channel, which directly leads to a higher success rate and better overall usability when compared to the Input channel variant that was implemented in the original BEDA paper. However, as the results of the user study are based on a fairly small data set of only fifteen participants, these conclusions should be interpreted more as a general tendency rather than hard facts.

Different people have different natural preferences for the vibration or the visual button channels. It is therefore important to provide different pairing methods and let the user choose his favorite channel. On mobile phones which support all different types of button channels, this could for example include the Short Vibration and Traffic Light channels as well as the Input channel.

A general limitation of all button-based auxiliary channels is the quite long duration of about 1 to 1.5 minutes for successful pairing. In comparison, other authentication methods based on audio or visual (bar code and camera) channels only take about 20 to 40 seconds [7]. However, these methods are only applicable if the devices meet the respective hardware requirements.

6.2 Future Work

At the moment, the capacity of all button channels is fixed to three bytes and the number of button events is fixed to nine events. With this setting, the pairing process takes about 1 to 1.5 minutes and guarantees an attack probability as small as 2^{-24}. However, depending on the use case, a user might be unwilling to spend this amount of time and might accept a higher attack probability instead. Some participants of the user study mentioned in the post-test discussion that they either found the pairing time too long or the number of button presses too high. It would therefore be an interesting question how a user is willing to trade off security against usability. This could be achieved by providing the possibility to the user to set the number of button presses before using a given channel. Depending on the chosen number of button presses, he would get an indication about the security level of the channel (like small, medium or high).

A User Study

A.1 Task Order

Participant	First Task	Second Task	Third Task	Fourth Task
1	Power Bar	Input	Traffic Light	Flash Display
2	Traffic Light	Flash Display	Input	Short Vibration
3	Long Vibration	Flash Display	Traffic Light	Input
4	Short Vibration	Power Bar	Long Vibration	Traffic Light
5	Flash Display	Input	Short Vibration	Power Bar
6	Long Vibration	Power Bar	Flash Display	Input
7	Flash Display	Long Vibration	Input	Short Vibration
8	Power Bar	Traffic Light	Short Vibration	Input
9	Input	Power Bar	Traffic Light	Long Vibration
10	Traffic Light	Long Vibration	Input	Short Vibration
11	Input	Short Vibration	Long Vibration	Power Bar
12	Power Bar	Traffic Light	Short Vibration	Flash Display
13	Flash Display	Traffic Light	Power Bar	Long Vibration
14	Long Vibration	Short Vibration	Flash Display	Traffic Light
15	Short Vibration	Long Vibration	Power Bar	Flash Display

Table A.1: Predefined task order for each of the 15 participants of the user study

A.2 Background Questionnaire

Benutzerumfrage

Demografischer Hintergrund

Alter
☐ 18-24 ☐ 25-29 ☐ 30-34 ☐ 35-39 ☐ 40-49 ☐ 50+

Geschlecht
☐ männlich ☐ weiblich

Schulbildung
☐ Obl. Schule ☐ Gymnasium ☐ Hochschulabschluss (Bachelor / Master) ☐ Doktorat

Erfahrung mit mobilen Geräten

Ich besitze ein mobiles Gerät wie z.B. ein Mobiltelefon, PDA oder Smartphone.
☐ ja ☐ nein

Ich habe schon Daten (z.b. Fotos) zwischen zwei solchen Geräten (inkl. Laptop) mit Hilfe einer Bluetooth- oder Infrarotverbindung übertragen.
☐ ja ☐ nein

Falls ja: Ich habe die Geräte zuvor gekoppelt (z.B. durch Eingabe eines PINs)
☐ ja ☐ nein

Figure A.1: Background questionnaire in German

A.3 Post-test Questionnaire

Bewertung der getesteten Kopplungs-Methoden

Ich fand, das Benutzen der Methode “gleichzeitig Drücken” war
□ sehr einfach
□ einfach
□ schwierig
□ sehr schwierig

Ich fand, das Benutzen der Methode “blinkender Bildschirm” war
□ sehr einfach
□ einfach
□ schwierig
□ sehr schwierig

Ich fand, das Benutzen der Methode “Verkehrsampel” war
□ sehr einfach
□ einfach
□ schwierig
□ sehr schwierig

Ich fand, das Benutzen der Methode “Fortschrittsbalken” war
□ sehr einfach
□ einfach
□ schwierig
□ sehr schwierig

Ich fand, das Benutzen der Methode “kurze Vibration” war
□ sehr einfach
□ einfach
□ schwierig
□ sehr schwierig

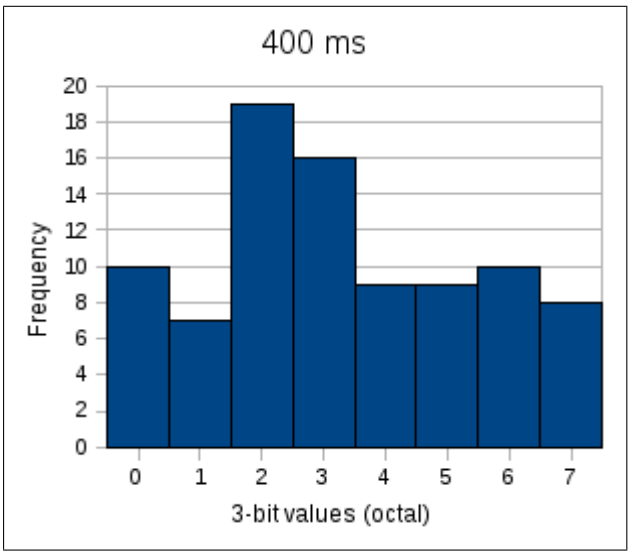
Ich fand, das Benutzen der Methode “lange Vibration” war
□ sehr einfach
□ einfach
□ schwierig
□ sehr schwierig

Von allen getesteten Methoden bevorzuge ich (mit Begründung):

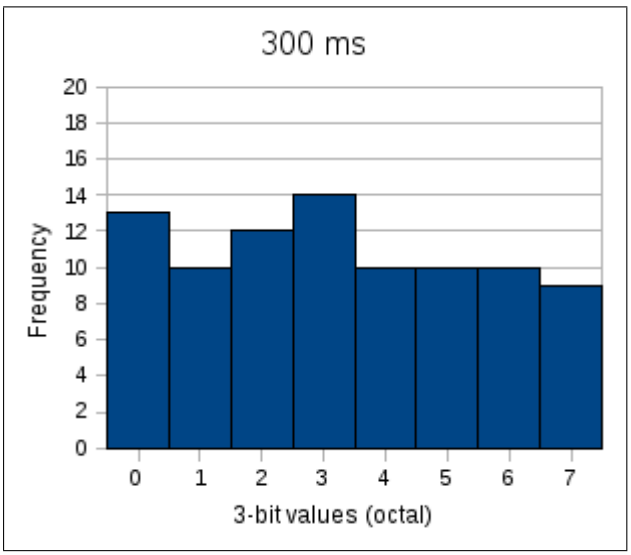
_____

Figure A.2: Post-test questionnaire in German

A.4 Statistical Analysis



(a) Candidate secret 1 (based on $t_{min} = 400$ ms)



(b) Candidate secret 2 (based on $t_{min} = 300$ ms)

Figure A.3: Frequencies of the captured 3-bit values for the Input channel

During 11 protocol runs with the Input channel, a total of 88 3-bit values have been measured (represented as octal values 0 to 7). The 3-bit values for candidate secret 1 were derived by rounding the captured intervals to 400 ms, those for candidate secret 2 were derived by rounding to 300 ms.

χ^2-test for candidate secret 1

Octal value i	0	1	2	3	4	5	6	7
Frequency f_i	10	7	19	16	9	9	10	8

Null hypothesis H_0: The sample set is uniformly distributed.
Alternative hypothesis H_1: The sample set is not uniformly distributed.

Degrees of freedom $df = 7$
Expected frequency $f_e = \frac{88}{8} = 11$

$$X^2 = \sum_{i=0}^{7} \frac{(f_i - f_e)^2}{f_e} = 11.27$$

$$\chi^2_{0.95}(7) = 14.06 > 11.27$$

With 95% statistical confidence, H_0 cannot be rejected.

χ^2-test for candidate secret 2

Octal value i	0	1	2	3	4	5	6	7
Frequency f_i	13	10	12	14	10	10	10	9

Null hypothesis H_0: The sample set is uniformly distributed.
Alternative hypothesis H_1: The sample set is not uniformly distributed.

Degrees of freedom $df = 7$
Expected frequency $f_e = \frac{88}{8} = 11$

$$X^2 = \sum_{i=0}^{7} \frac{(f_i - f_e)^2}{f_e} = 2.00$$

$$\chi^2_{0.95}(7) = 14.06 > 2.00$$

With 95% statistical confidence, H_0 cannot be rejected.

Acknowledgments

I would like to thank Prof. Srdjan Capkun for supervising my thesis and giving me the opportunity to write my thesis on an interesting and active research topic. I would like to express my gratitude to Iulia Ion for her support and the many great ideas that influenced this thesis. I also want to thank Rene Mayrhofer, the founder of the OpenUAT project, for answering my questions during the implementation phase of this thesis.

Bibliography

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[2] Christian Gehrmann, Chris J. Mitchell, and Kaisa Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7(1):29–37, Spring 2004.

[3] Michael T. Goodrich, Michael Sirivianos, John Solis, Gene Tsudik, and Ersin Uzun. Loud and clear: Human-verifiable authentication based on audio. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 10. IEEE, 2006.

[4] Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In *The 5th International Conference on Cryptology and Network Security, CANS 2006*, volume 4301 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2006.

[5] Rene Mayrhofer. Towards an open source toolkit for ubiquitous device authentication. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 247–254. IEEE Computer Society, 2007.

[6] Rene Mayrhofer and Hans Gellersen. Shake well before use: Authentication based on accelerometer data. In *Proc. Pervasive 2007: 5th International Conference on Pervasive Computing*. Springer-Verlag, May 2007.

[7] Rene Mayrhofer and Iulia Ion. OpenUAT: The Open Source Ubiquitous Authentication Toolkit. 2009. *to appear*.

[8] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *IEEE Symposium on Security and Privacy*, pages 110–124, 2005.

[9] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. `http://www.cacr.math.uwaterloo.ca/hac/`.

[10] Donald A. Norman. *The design of everyday things*. Basic Books, New York, 2002.

[11] OpenUAT. The Open Source Ubiquitous Authentication Toolkit. `http://www.openuat.org`.

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

[13] Jeffrey Rubin and Dana Chisnell. *Handbook of Usability Testing, Second Edition: How to Plan, Design, and Conduct Effective Tests*. Wiley Publishing, 2008.

[14] Claudio Soriente, Gene Tsudik, and Ersin Uzun. BEDA: Button-Enabled Device Pairing. In *Proc. IWSSI 2007*, pages 443–449, September 2007.

[15] Claudio Soriente, Gene Tsudik, and Ersin Uzun. Hapadep: Human-assisted pure audio device pairing. In *ISC '08: Proceedings of the 11th international conference on Information Security*, pages 385–400, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] Ersin Uzun, Kristiina Karvonen, and N. Asokan. Usability Analysis of Secure Pairing Methods. In *Financial Cryptography*, pages 307–324, 2007.

[17] Ford-Long Wong and Frank Stajano. Multi-channel protocols. In *Proc. Security Protocols Workshop*. Springer, 2005.