NTNU – Trondheim
Norwegian University of
Science and Technology

Engineering device pairing with fuzzy cryptography

Sinisa Matetic

Master in Security and Mobile Computing
Submission date: June 2014
Supervisor: Colin Alexander Boyd, ITEM
Co-supervisor: Tuomas Aura, Aalto University

Norwegian University of Science and Technology
Department of Telematics

Title: Engineering device pairing with fuzzy cryptography

Student: Sinisa Matetic

Problem description:

The goal of the thesis project is to analyze different kinds of device pairing methods that are based on unconventional methods. Additionally, the goal is to investigate a novel type of device pairing using the contextual information given from synchronized drawing with two fingers on two different devices. This device pairing procedure is to be analyzed in terms of potential metrics for drawing comparison and furthermore incorporating the idea into a new protocol design based on fuzzy cryptography.

Responsible professor: Tuomas Aura, AALTO University,

Tuomas Aura, AALTO University

Supervisor: Colin Boyd, ITEM, NTNU

Colin Boyd, ITEM, NTNU

Abstract

Device pairing protocols are a subset of secure communication protocols used to bootstrap a secure channel over an insecure communication link between two or more devices. Example protocols use technologies such as Bluetooth or infrared light and are mostly based on user-entered secret keys or secrets directly verified and authenticated manually by users. However, in this thesis we focus on four different areas that complement the existing protocols. Firstly, we overview protocols that are based on fuzzy secrets and that utilize contextual information to pair device. Secondly, we analyze a particular method that uses contextual information, synchronized drawing with two fingers of the same hand on two touch screens or surfaces, to derive a shared secret by applying various metrics and conducting measurements and comparisons. The main results from this parts are new, improved metrics for comparing fuzzy secrets that consist of a drawing or movement path Thirdly, we overview the mathematical constructions that support fuzzy cryptography schemes and describe our own system architecture based on these. Fourthly, we develop a secure device pairing protocol based on synchronized drawing that uses fuzzy cryptography and error-correction codes in order to derive a shared secret between devices that share similar, but not exactly the same, secret noisy inputs. While the protocol is based on an information-theoretically secure construction, we find that the security of the practical implementations is harder to prove because of uncertainty about the amounts of entropy in the shared noisy inputs. The protocol nevertheless has the characteristics of practical security. Additionally, we describe information-theoretically secure alternatives derived from available theorems in the literature.

Preface

This Master Thesis is carried out as a joint project between Aalto University, Espoo, Finland as the main university and Norwegian University of Science and Technology, Trondheim, Norway as host university. The author would like to thank his main supervisor, prof. Tuomas Aura, for his direct guidance, help and support during work on this project. Additionally, the author would like to thank his co-supervisor, prof. Colin Boyd for his assistance and advice given in the host university.

"Neka ti uvijek bude na umu da samo stvaralacki rad stvara fizionomiju licnosti". A.B.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

2PCP Two Phase Commit Protocol.

CDF Cumulative distribution function.

DH Diffie-Helman.

ECD error-correcting distance.

FNR False negative rates.

FPR False positive rates.

ICDF Inverse cumulative distribution function.

IR infrared.

IRC Infrared communication.

IT Information Technology.

LED Light-emitting diode.

MDS Maximum distance separable.

MitM Man-in-the-Middle.

OOB Out-of-band.

OS Operating System.

PKI Public key infrastructure.

RS Reed-Solomon.

SSL Secure socket layer.

TCOT Transaction Commit On Timeout.

WiFi Wireless fidelity.

Chapter 1
Introduction

Today we are faced with the fast emerging of various electronic devices and technologies used for communication between people and devices. In order to satisfy the needs of the user population in situations where they want to communicate and share information we need to find a suitable way for their devices to pair. In this thesis we focus our work towards investigating properties and designing secure device pairing protocols. These protocols may be seen as a subset of secure communication protocols in general which are used to create a secure channel over an insecure network (mostly wireless) between two or more device that want to affiliate with each other. General motivation related to this topic is based on the need of defining more secure protocols for various device pairing since the number of electronic devices is exponentially growing, especially in the form of personal electronics (mobile phones, tablets, laptops, various accessories). The second influential factor is the need to create pairing protocol that benefit from ease-of-use characteristics since the currently used protocols are not quite simple and mostly require several manually performed procedures. Some novel ideas where tried and implemented on the lower usage level to test their feasibility. Specifically, this work continues on the work of Sethi et al. [SAA13] who firstly introduced the idea of a commitment based protocol based on synchronous drawing on two different devices.

1.1 Goals

The starting point was to survey the existing literature on the topic and summarize the findings in a concise and easily-understandable taxonomy. Throughout the analysis we focused naturally more on the ideas based on fuzzy cryptography since their implications were still underused and provide an interesting approach to the problem. The idea and hence the main research question in this thesis was to analyze the possibility of designing a secure device pairing protocol that bases its secrets on contextual information available to the devices, in specific, synchronized drawings on touch-sensitive surfaces. This implies the need of defining several more secondary

research questions: i) implications related to encoding the drawings captured from different devices, ii) metric that allow effective comparison of different drawings made by the same or different users, iii) mathematical background that supports the implications of fuzzy cryptography, and lastly iv) feasibility of creating a protocol under given circumstances defined by the protocol characteristics.

These questions were answered throughout the thesis and are mostly oriented to specific chapters of the thesis. The methodology we used in conducting the research was carefully planned and systematical organized.

1.2 Outline

The thesis is structured and formatted in the following way:

1. Firstly, we focus on investigating various approaches, used previously, and related to device pairing protocols in a form of a literature overview presented in Chapter 2, *Background*,
2. We continue to shift our focus to a specific protocol type based on synchronous drawing and we analyze its implications along with different encoding option and comparison metrics that could be use to bootstrap secure communication (Chapter 3, *Metrics for synchronous drawing*),
3. Naturally, the research continues on the mathematical implications related to the protocol design which are mainly focused on various construction that allow usage of contextual fuzzy inputs, and fuzzy cryptography in general (Chapter 4, Introduction to fuzzy cryptography and component constructions),
4. The last main chapter involves the actual construction of the device pairing protocol and the analysis of its characteristics, both theoretical and practical (Chapter 5, *Protocol design*), and
5. We conclude our work and summarize the findings in the last chapter (Chapter 6) where we also present the ideas for future work on the topic).

Chapter 2
Background

Device pairing protocols are a subset of secure communication protocols used to bootstrap a secure channel over an insecure communication line between two (or more) devices that were previously unaffiliated [KSTU09]. These protocols are characterized by the lack of well accepted certification or trust infrastructure which implies specific user involvement in the pairing process. In this thesis we use the term *pairing* to emphasize the initiation process of establishing the needed secure communication over a wireless channel. While the number of different methods to perform and establish a security association vary in regards to the human-perceptible channels, they all have the same characteristic - vulnerability (or at least openness) to Man-in-the-Middle (MitM) attacks.

The increasing and rapid dissemination of personal mobile electronic devices that use wireless communication is making the *problem* of secure device pairing even more important and urgent [UKA07]. Short-range wireless communication technologies such as Bluetooth, ZigBee, Wireless fidelity (WiFi) are and will remain popular in the future evolution of new services offered by various personal devices. The most common usage scenarios are connecting two different phones to exchange data or cooperate through some service, connecting wireless headphones with mobile phones or maybe accessing wireless printer from a range of devices.

As mentioned before, the diversity of devices makes it extremely difficult to standardize certain protocols that would fit a wide range of equipment. Thus, forming a global infrastructure may not be feasible in the current development stage. This issue regarding secure key exchange has drawn a lot of attention both by researchers and experts [KFR09]. Bluetooth protocol for device pairing has been identified with security weaknesses [JW01] and new protocol propositions are emerging [CCH06][LN06a][NR06][Vau05]. Currently available and applied protocols provide reasonable security but are subject to explicit user interaction relatively unnatural and uncommon for non-Information Technology (IT) persons. Hence, in the past several years the focus has turned to pervasive computing research in search

for more natural approach of pairing various devices [SAA13].

This thesis focuses exactly on extending research on device pairing protocols in the environment of ubiquitous computing. We focus our research on protocols that use contextual information available by monitoring sensors of aimed electronic devices. The bootstrapping of the secure channel is based on a *fuzzy* secret that different devices acquire separately by observing the same environment. However, the term *fuzzy* implicates that those inputs from different devices are not identical, yet approximately similar. The fuzzy secret or the shared sensory input may be extracted from various contextual information, such as shaking of devices with a gyroscope and accelerometer [May07][MG07][GM12][KSW07][BSHL07], ambient audio signals and radio frequencies [STU08][SS13][MMV+11][NSHY12][NSHJ12][SJ12] or making usage of the device's camera for authentication [MPR05][MW07][SEKA06].

In this chapter we provide a survey of existing methods for bootstrapping secure channels between two devices while focusing on authenticated key-establishment methods based on fuzzy inputs and approximate verification. Alike [SAA13] we consider protocols that are able to tolerate errors on the bit level data used between two devices to extract the shared secret.

2.1 Taxonomy of device pairing methods

The taxonomy of different device pairing methods that we describe follows the one proposed in [SAA13] while we add/remove some of the methods, and some are more closely granulated. Figure 2.1 illustrates the proposed taxonomy where we concentrate more on the device pairing methods bound to fuzzy data.

2.1.1 Out-of-band channels

As commented by [KSTU09], a popular research direction regarding device pairing method is the utilization of Out-of-band (OOB) channels (category 1 on Figure 2.1). OOB channels are perceivable by human users that initiate the pairing of devices and thus provide a method to use human-sensory capabilities in order to authenticate human-imperceptible information transferred through other channels, such as a wireless link used for almost all device pairing protocols. OOB channels are in general secure, since an active/passive attacker will be detected upon interfering with the bootstrapping process [KFR09]. However, the level of security greatly depends on the users which should perform critical tasks, related to protocol design, correctly. The simplest example of the OOB channel may be perceived as direct human communication which directly implies an establishment of a certain level of trust.

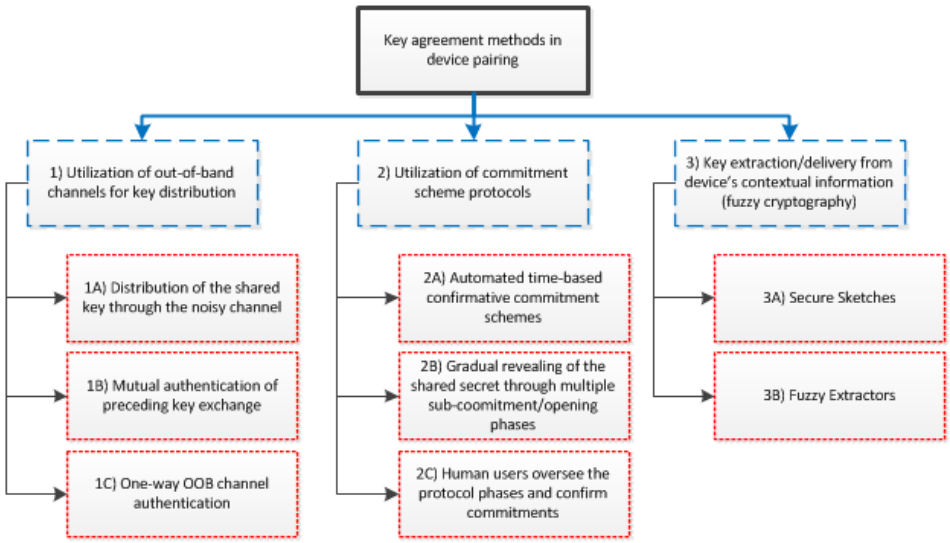


Figure 2.1: Taxonomy of different device pairing methods (follows loosely [SAA13])

Figure 2.2 illustrates the usage of an OOB channel. Symbols situated on the top line represent camera, infrared, audio and light, respectively, as methods for transferring messages through an OOB channel. As mentioned before, authentication and integrity are the key values of the OOB channel, while secrecy depends on the exact environment. Hence, humans (or devices) interacting over the channel may be assured that the communication link is not modified or tampered with (under the assumption that the environment is under control of the users). However, eavesdropping may not be completely disregarded since the environment in which the pairing takes place almost always has more *participants* that may act as potential attackers. Yet, *they* can only listen to the information exchange and not block or modify it without directly being detected. This way of communication makes the OOB channel an excellent candidate for authentication and integrity verification of messages exchanged over a normal (e.g. wireless) channel.

From the explanation above we may easily differentiate two main types of usage of the OOB channel. Firstly, the OOB channel itself may be used for direct key distribution over a noisy channel, and secondly, the OOB channel is only used for authentication. An example of the first solution is presented in [STU08] where the user monitors an audio channel used to send an encryption key with error correction. The second solution for verifying result of a key exchange is more common.
For example, if we presume the existence of a public key infrastructure (PKI) in device pairing protocols, two devices could exchange their public RSA keys along with other information over a wireless channel, compute some individual cryptographic

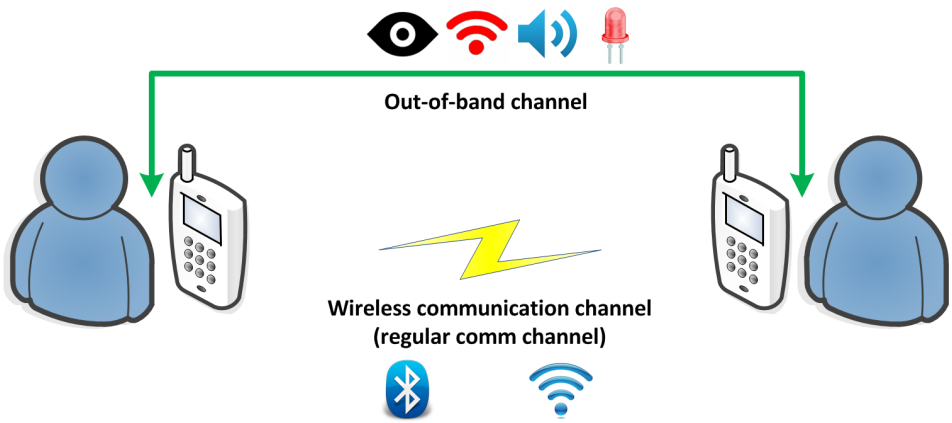


Figure 2.2: Out-of-band channel and technologies used for transfer

hashes (or digests [KFR09]) that would be verifiable through the OOB channel and some human interaction. The verification itself may take several forms; two device display information (e.g. short strings) that a human user can compare, or one device shows a string and the human user enters that exact string to another device. Upon successful authentication over OOB channel, the public keys are authenticated alike. Thus, device can bootstrap a secure communication channel (e.g. Secure socket layer (SSL) connection) for subsequent message exchange.
These methods for using OOB channels for device pairing have received a lot of attention in the research community. Hence, several proposal have been developed and we summarize the most important ones here by loosely following the list from [KSTU09]:

1. "Talking to strangers" was an early method developed by Balfanz et al. [BSSW02] that uses Infrared communication (IRC) as the OOB channel. Even though the method has been significantly tested, there are some drawbacks regarding usage of infrared (IR) channels. Firstly, the method requires alignment of two devices, more specifically, alignment of IR sensors, which might not be so trivial for non-technical users, and a direct line of sight between device. Additionally, infrared (IR) channels are vulnerable to MitM attacks which makes the protocol design vulnerable as well. Nowadays, there are only few devices equipped with IR communication capabilities since they are replaced with more sophisticated technologies (e.g. Bluetooth),
2. "Resurrecting Duckling" represents the initial attempt of creating a device pairing protocol that would completely mitigate the MitM attack [SA02]. The protocol truly does its purpose but it requires physical interfaces and a cable connection between devices. Hence, the sole purpose of wireless pairing is defeated and the pairing itself is troublesome and therefore it it archaic today.

However, in the early 90s it was appropriate.

3. Image comparison is also a popular method that has been present from the beginning. The data received through OOB channels is encoded and the users are asked to compare the images (this however implies more involvement of the users). Some appropriate examples surveyed in [KSTU09] include creation of a 25 bit-per-color flag used for subsequent comparison [ED03], *hash visualization* and comparison [PS99], "Snowflake" [Gol96]. However, such protocol version implies the need of high resolution displays (which today does not pose a serious problem since we are experiencing a drastic evolution of display technologies over the last decade),
4. Recent protocol proposals include the usage of a camera. While this poses more restrictions on device innovation and requires more high-end products, the diversity of options to exploit various OOB channels grows. For example, "Seeing-is-believing" [MPR05] combines two different approaches which requires a display on one device and an unidirectional visual OOB channel (camera) on another device. The first device encodes OOB data into a two-dimensional bar code that is then displayed on the screen while the second device's camera is then used to record the picture and process it. Another protocol involves usage of only one camera or light detector (i.e. visual receiver), while the other device has to have a Light-emitting diode (LED) to transmit OOB data by blinking [SEKA06]. The receiver stores the pattern, extracts the information from the inter-blinking gaps and lastly decides about the successfulness of the protocol session.
5. Extensive research has been conducted towards using audio signals as well. An example is HAPADEP [STU08] that is solely based on the audio communication between two devices. Both devices need to be equipped with a microphone and a speaker. All crypthographic messages are exchanged over audio, and the user's responsibility is to only strictly monitor "interaction for any extraneous interference" [KSTU09] (example of a category 1A protocol in Figure 2.1). Some variations of the protocol have been made where the audio channel is backed up with the wireless channel (lacks in HAPADEP). Thus, audio is only perceived as an OOB channel. Another method relevant to mention is the "Loud and Clear" [GSS$^+$06] audio pairing mechanism. It may be perceived in two different settings where its either pure audio-audio (speaker-speaker) or a combination of audio and visualization (speaker-display). Namely, the first setting implies that a user compares two vocalized sentences while in the second setting user is bound to compare a sentence from a display with the sentence heard from the speaker.

Many of the previously mentioned protocols and device pairing methods are based on the idea of S. Vaudenay [Vau05]. The basic idea is to establish peer-to-peer communication over an insecure channel that is authenticated. To succeed, an

extra channel is used that may authenticate short strings (10-15 bits) that exerts the "weak notion of authentication" ([Vau05]) to eliminate the possibility of string forging/modification, but on the expense of possible denial-of-service and replication attacks. The proposed protocol itself mainly relies on the commitment scheme methods which we discuss in the next subsection.

2.1.2 Commitment-based protocols

Commitment-based protocols are a subset of device pairing protocol that combine widely known and adopted key establishment methods with separate authentication of the latter. Most commonly used protocols for unauthenticated key-exchange are Diffie-Helman (DH)[DH76] or public-key encryption schemes without certificates (and in that sense without certified Public key infrastructure (PKI)[1]) [SAA13]. Thus, the resulting key shared between devices is strong but not yet authenticated which makes the protocol layout open to various spoofing methods and the MitM attacks.

According to Balfanz et al. [BSSW02] and Vaudenay [Vau05] the solution for securing communication over insecure channels may go to two opposite directions: i) remove the confidential channel (eliminates the need for explicit key generation with the usage of public keys), or ii) use short passwords rather than long secret keys. However, the best possible solution lies in the combination of the two previously mentioned directions where an extra channel is used only for authentication (and is able to transmit only short bitstrings of data). Hence, a regular key agreement protocol (such as DH) may be strengthened by authenticating all of the exchanged messages.

One of the initial protocols that used commitment-based bootstrapping and aimed to protect against MitM attacks was the interlock protocol by Rivest and Shamir [RSA78][RS84]. The principal of that old protocol is much like what we may see today with the exception of focusing on encryption rather than using collision-resistant hash functions[2] that are common today.

The next several subsections introduce the principles of the commitment scheme, survey the distinct authentication methods for the key agreement and lastly show some examples of the usage of commitment scheme in computer and information systems.

[1] Ad-hoc networks are not able to assume the existence of a centralized third party to vouch for setting up a secure network among users [Vau05]

[2] A cryptographic hash function is a hash function that takes an arbitrary block of data and returns a fixed-size bit string, the cryptographic hash value, such that any (accidental or intentional) change to the data will (with very high probability) change the hash value [online source: wiki]. "The property required from the hash function h is that for a given value x it is computationally hard to find a y such that $h(y) = h(z)$ and $y \neq x$" [NY89].

Principles of the commitment scheme

The commitment scheme can easily be explained through its four phases:

1. As mentioned previously, most of the protocols based on commitment schemes start with the unauthenticated key exchange. Let us presume now the existence of a strong shared key k that can be used for encrypting communication. This fact is the result of the first protocol phase,
2. The second phase begins by exchange/distribution of another shared secret w (usually short, ranging from 4-6 digits) delivered over an out-of-band channel, and we call this *commitment phase*. This phase is carried through the users' mutual commitment to a certain value. Each of the users generates/picks a random number x that has the purposes of masquerading the original secret, and upon that computes a cryptographic hash $H(k, w, x)$ called the *commitment*,
3. In the third phase the users exchange the calculated commitments and the next and final phase of the protocol may begin, and
4. The fourth phase is called the *opening phase* where both of the parties reveal to each other the chosen random numbers (x). Now both of them may recompute hashes to verify the trustworthiness of the opposite party [Nao91].

If we take a closer look at the design of the commitment scheme we may observe that its security is highly time-dependent. Namely, the critical moment that opens a possible attack for the adversary is between the commitment phase and opening phase. Both of the parties in the protocol *have* to receive commitments before they reveal their generated random number x. If this schedule is not followed a MitM attacker may find the value of the short shared secret within reasonable time to take advantage of the protocol run (simply by brute-forcing through all the combinations).

The research community proposes several different ways to overcome this issue which are all relevant and case specific:

– The most easiest and logical way to impose the time order is to entrust users to oversee the protocol phases and manually confirm the completion of the commitment phase (on both of the device) before starting the opening phase (category 2, subcategory C in Figure 2.1). However, users tend to be impatient and not trained accordingly to perform the protocol run. In that sense, they might accept the commitment phase on one device before it is done on the second device (not yet even able to confirm the completion). The first device immediately starts the opening phase and reveals it random generated number x. As discussed by Sethi et al. [SAA13] it might be difficult to design a proper user interface where the users could not confirm the completion of the commitment phase without actually checking on both devices that it is done[3].

[3]One possibility is to display a warning message before the user is given the opportunity to confirm the phase, but as discussed, general users without technical backgrounds tend to skip the *boring* instructions and continue with the quickest path to their goal

– The second way of imposing the time order it to automatize the process of secret revealment. Hence, this can also be done in two ways. Firstly, the protocol can predict the time needed for both commitments to be received up to a certain level of certainty (e.g. 500 ms). However, the protocol designers should be careful when choosing this solution since the decision about the time frame might be difficult to realize. In one way, a short period of time might make the protocol vulnerable if some problems in the communication arise, and in another, if the period is too long, the users have to wait more which lowers the user experience level. Both of these solutions and the proposed user involvement fall in the category 2, subcategory A in the Figure 2.1. A second solution (category 2, subcategory B) is to divide the second and third phase of the commitment protocol into subphases by gradually revealing the shared secret receive through the OOB channel. In that sense, the shared secret w is presented as $w = w_1w_2w_3...w_n$ where n is the number of the protocol subphases. For each of the subphase a separate commitment is calculated and the relevant part of the secret is revealed subsequently. A MitM adversary may try to spoof one of the subcommitments H_i and thereby learn and replay a part of the shared secret w_i. Yet, the attacker is blocked after continuing to the next phase if the spoofed commitment proves to be incorrect. In this way, users can detect the misuse of the protocol, abort the session and re-initiate the key exchange/authentication [GN01][UKA07][SVA07].

Taxonomy of key-establishment authentication schemes

As discussed previously, the key establishment/agreement may be based either on asymmetric or symmetric cryptography. In both cases after the key has been distributed or derived we may proceed to the authentication process. However, there are protocol versions that are unauthenticated and those are vulnerable to active attacks (e.g. MitM) in case of asymmetric crypto or to passive (e.g. plain eavesdropping) and active attacks in case of symmetric crypto.

Distinct authentication procedures have been surveyed by Suomalainen et al. [SVA07]. Figure 2.3 illustrates the possible authentication options depending on the protocol design. Authentication in the asymmetric key agreement protocols may be divided into three different categories [SVA07]:

1. **Authentication based on shared secrets** implies usage of short pre-shared secret passkey P that is known to both devices. The secret passkey distribution might be done in several ways, such as actively using human users to choose and enter the same key to both devices, or one device might generate a code that needs to be manually entered to another device (see Bluetooth characteristics). Another option is the usage of OOB channels for passkey delivery (this option

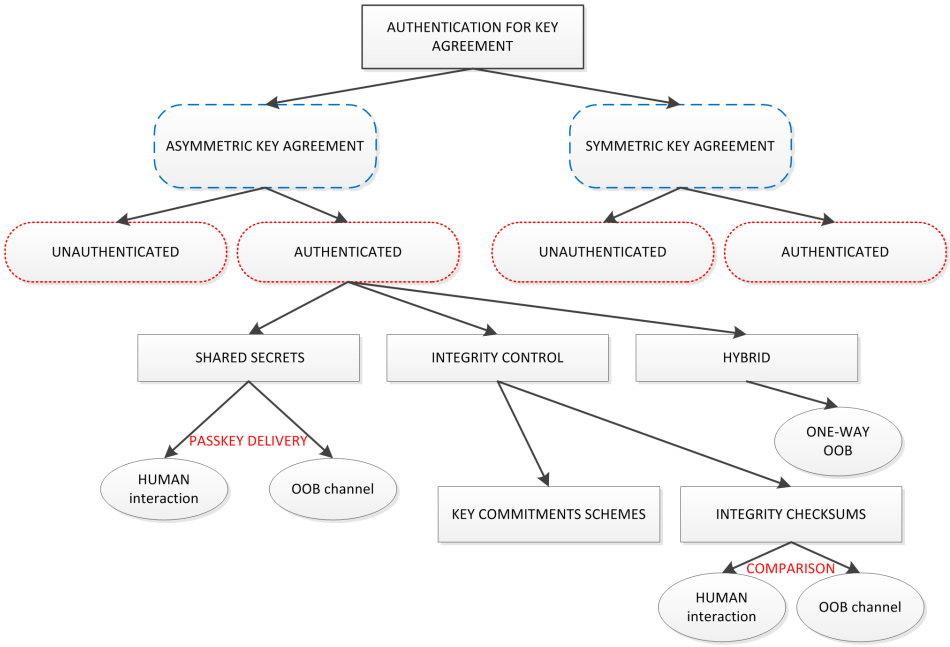


Figure 2.3: Taxonomy of authentication methods

is widely used and studied in the research community and we have already discussed it in Subsection 2.1.1).

After choosing the right method of passkey distribution there are several methods available for the actual authentication. A good example is the time-based commitments described previously which was originally proposed by Gehrmann et al. [GMN04]. Gehrmann propose three different schemes, MANA I (MAN-ual authentication), MANA II and MANA III where the latter one presents a time-based commitment. The idea is in the separation of the secret passkey P into k parts where the secret is then gradually revealed[4]. From there, many different MANA III-alike protocols have been proposed and developed [BM92][VSLDL07].

2. **Authentication through integrity control** may be divided into two separate methods: i) authentication based on key commitments [BSSW02], and ii) authentication based on integrity checksums. The former implies usage of OOB channels to deliver commitment which might be either public keys of users (devices) or their hashes[5].

   The latter method implicates computation of short checksums from the preced-

[4]This was described earlier when we discussed various commitment based alternatives

[5]Note that these commitments need to be of enough size to resist a possible search of a second pre-image by the adversary.

ing messages exchange of the key agreement protocol. The comparison of the checksums results in the authentication decision. This method of authentication has been discussed and proposed by several researchers [LN06b][PV06][CCH06], but may be formally referred ([SVA07]) as *short authenticated string* originally proposed by Vaudenay [Vau05]. One simple example of this protocol is that the public keys are exchanged and a cryptographic hash function $f()$ is created in order to map the short output of the hash into human-readable strings that are then compared by users directly. Involving users to do the comparison of checksums is one way of the last protocol step, while the alternative may be to use a physical OOB channel.

3. **Hybrid authentication** implies authentication protocols that achieve the means of mutual authentication when the channel and/or devices are limited in their capabilities. More specifically, the method is used when a one-way OOB channel is available. This channel is used to transmit some secret value and a public key hash from one device to another. The receiving device easily authenticates the first device based on the public key hash, while the sending device conducts authentication by checking that the receiving device has knowledge about the shared secret key.

Regarding authentication in the symmetric key agreement the most usual method is by using a sufficiently long *pre-shared* secret so the offline/online computation are not feasible in reasonable time-frames for the protocol to be securely bootstrapped.

Examples of commitment scheme protocols

The usage and application of commitment schemes is more broader than just device pairing protocols. Berger et al. [BH03] focuses studies on the extended asynchronous versions of the *pi*-calculus for distributed computing representation. The aim was to examine "their descriptive power to the description and correctness proof of an important distributed algorithm" [BH03], the Two Phase Commit Protocol (2PCP)[6], as they call it. Another example involves the development of transaction protocols, more specifically a timeout-based mobile transaction commitment protocol called Transaction Commit On Timeout (TCOT) [KPDS02]. The idea focuses on the timeout approach (highly relevant to pairing protocols) that is used to reach a "final transaction termination decision in any-message oriented systems". This approach solves issue often experienced in wireless environments where the link might be slow and/or unreliable. Chopra et al. [CS06] claims that the commitment protocols and their preferences are to be contextualized so that in different settings they are able to transform themselves accordingly[7].

[6]The 2PCP is a distributed protocol which consists out of several process, possibly faulty, that interact via possibly faulty channels [GR93][BH03]

[7]Even thought the contextualization and transformation is not bound to device pairing protocols, the idea is relevant to the commitment scheme design since we use contextual information in

However, the idea of bit commitment protocol presented by Naor [Nao91] might be recognized as the predecessor of the commitment schemes that we discuss here. As shown in the original paper, a pseudorandom generator may be used to provide a bit-commitment protocol where two users choose a bit b and put it "*into a locked box*" by each of the users. The boxes are exchange and opened afterwards to prove trustworthiness and authentication. These kind of schemes were historically used in zero-knowledge protocols [GMW86], multiparty protocols [GMW87][CDVdG88] and identification schemes[8] [FS87].

2.2 Fuzzy contextual data for key establishment

Cryptography applications naturally rely on uniformly distributed and random vectors to be the secrets with the properties of precise reproduction. However, in some cases it may be difficult to create, store and retrieve such vectors. In applications related to fuzzy cryptography the vectors are often neither uniformly random nor reliably reproducible. One of the examples where fuzzy cryptography is naturally used, and which proves the point, is a random person's fingerprint or iris scan that is clearly not a uniform random vector and it does not reproduce precisely each time it is measured. Additional applications consider long password phrases, answers to many security questions ([FJ01]) or even lists of favorite objects from a certain set ([JS06]) that share the properties of biometric measurements as well with the addition that even though they are human friendly it may be difficult for a human user to remember them.

Biometric measurements in general seem to contain more entropy than human memorizable passwords. However, due to its characteristics, two readings are almost never the same, yet they are likely to be close and similar. In the same fashion, humans are unlikely to remember answers to all of their security questions and some of the answers may be presented in similar forms. Thus, fuzzy cryptography mechanisms introduce the ability to tolerate a specified number of errors in the security vector while retaining security remains crucial if we are to increase and achieve better security properties than the most common and typical user-chosen passwords.

Two of the most applicable constructions for fuzzy cryptography schemes may be viewed as secure sketches and fuzzy extractors (our protocol design is based on the combination of those and they are discussed in detail further on in the thesis) as introduced coherently by Dodis et al. [DRS04]. More specifics about those can be found in Chapter 4 about mathematical background for constructing fuzzy device

bootstrapping the secure channel over the OOB channel

[8]Note that the original paper of the bit-commitment dates to 1991 which makes these protocols outdated and old. Yet, the proof of a concept is present

pairing protocols, while some real applications may be found as a web source by Harmon et al. [Har60]. In short, secure sketches and extractors may be viewed as fuzzy key storage functions where they allow the recovery of a shared secret (in the case of secure sketches we acquire the original fuzzy input ω from a device or human measurement, while in the case of fuzzy extractors we first use the fuzzy input on one side to create a secret key R and then on the other side using a generated helper vector retrieve that secret with the help of the second fuzzy input ω'). Additionally, in the standard setting of error-correction, when used in case of binary communication channels, the error tolerance is more higher when the errors are random and independent than when the errors are determined adversarially. Thus, it is important that *fuzzy* constructions meet the Shannon's bound ([SGB67]) for correcting random errors and also be able to fix the errors even if they are adversarial. Further reading on example constructions and the coding literature that supports it may be found in [HB01][MPSW05][Lip94][Zad97][Lan04][BBR88].

A concrete example for utilizing fuzzy cryptography and extractors in the case of password authentication involves a server, that stores the helper vector P and $f(R)$, where f is a one-way collision resistant function (e.g. hash). When the user inputs a similar ω' that is close to the original ω, the server can reproduce the original R using P and check if $f(R)$ matches the stored one. Since we assume that the random number R is $\epsilon - close$ to uniform, the adversary only has the option to revert $f(R)$ by the additive amount of ϵ. Thus, R may be used in various cryptographic applications such as symmetric encryption or generation of public-secret key pairs.

Another important example is the fuzzy commitment scheme protocol introduced by Jules at al. [JW99]. This article is the first one that introduces the "fuzzy" interpretation that may be used to derive secrets from noisy environments and then further on recover those secrets with vectors not completely the same as originals but ones that are close enough. The paper covers the mathematical constructions and tests for the error-correction mechanisms but the real world applicability is left out since it can be used in numerous areas. Juels et al. continues the research on fuzzy cryptography by publishing a paper under the title "Fuzzy vault scheme" [JS06]. At that time they introduce a simple and novel cryptographic construction that involves a player Alice that places a secret value x in a *fuzzy vault* and "locks" it using a set A of elements that are a part of some bigger public universe U. If a player Bob tries to "unlock" the vault using a similar length set B, he will obtain the secret value x if and only if A and B are close, more specifically in set difference metrics, if A and B overlap substantially. The authors prove that the fuzzy vault scheme is provably secure against a computationally unbounded attacker with the application to problems related to protecting data in numerous real world and error-coexistent environments. Such applications include user authentication systems, password recovery, biometric systems where readings are prone to be noisy due to the nature

of their scanning/recording procedure.

Background relations to other work

As mentioned before, error tolerance in biometrics was formally studied by Juels et al [JW99]. Less formal solutions that cover simple ways for error tolerance in uniformly distributed passwords were studied at the same time by Davida et al. [DFMP99] and Ellison et al. [EHMS00]. Moreover, extended solutions to these introduce entropy analysis which becomes relevant in error-correction systems [FJ01]. However, it is worth mentioning that similar approaches to error tolerance have been explored earlier in the cryptographic information reconciliation literature in the context of quantum cryptography where Alice and Bob try to acquire a secret key from secrets with small Hamming distances [BBR88][BBCS92].

In general, the need to deal with non-uniform passwords with low entropy has been realized long time ago in the security community with a range of different approaches. To increase the security of password authentication scenarios, Kelsey et al. [KSHW98] proposed usage of $f(\omega, r)$ instead of only password ω where r is a public random *salt* that makes it difficult for the adversary to perform a brute-force attack. However, this approach did not add any entropy to the password and as well did not imply the needed properties for the function f. A more applicable and realistic approach encompasses adding biometric features to passwords, where *biometric* information may be acquire through, for example, asking users to answer series of n personalized questions and thus using those answers to encrypt the actual secret [EHMS00]. A similar approach proposed by Monrose et al. [MRW02] uses the user's keyboard dynamics (and further one, voice [MRLW01]). Both of the mentioned examples require designing a secure fuzzy encryption and propose using heuristic designs (various forms of Shamir's "How to share a secret" [Sha79]) but do not address formal analysis and extensive research in the properties of error tolerance.

Some other approaches for guaranteeing the privacy have been considered. These are mostly related to the privacy of noisy data. Quantization for correcting errors via random physical functions was considered by Frey et al. [Fre01] while Barral et al. [BCN04] introduces systems for fingerprint comparison in offline and private mode. Privacy amplification research along with the research on derandomization and hardness amplification ([BBCM95] [HILL99]) addressed issues related to the extraction of uniform randomness from a random variable under the assumption that some information might have leaked. Along with these research topics, a major focus has also been put in the development of ordinary extractors (not fuzzy that we use in this thesis) with short seeds [Sha02].

With this section we finish the Chapter related to the background of commitment based and fuzzy device pairing techniques. Following-up is the metrics chapter that

explains the usage of synchronized drawing and its implications related to drawing comparison.

Chapter 3
Metrics for synchronous drawing

To create a novel type of a device pairing protocol that encompasses synchronized drawing on two separate device we need to define a specific metric for the inputs that can be used in the protocol design. The basic idea of the device pairing protocol is that the user wants to pair two devices that are equipped with a touch screen or another touch sensitive surface that can record movements. The user then aligns two drawing surfaces next to each other and draws the same picture synchronously on both devices with two fingers of the same hand. In this way the extent of similarity between two pictures is assumed to be high in comparison to drawing pictures with two different hands[1]. Typically, users choose to use the thumb and index finger, as illustrated in Figure 3.1, but some use the thumb with the middle finger as well. While drawing, the users can also see the lines on devices that have a screen (observable in the same Figure 3.1).

After the recording is done both of the devices have the input data in the form of a set of coordinates in the Cartesian coordinate system. The systems starting point (0,0) is in the lower left corner of touch surface (first pixel of the touchscreen). The coordinates of one captured event are accompanied with timing of when the event was recorded by the device's operating system (OS). We can call this information *raw data*. Hence, *raw data* of one drawing consists of a certain number of recorded events that depend on the duration[2] and movement speed[3].

This shared input is consequently used as the fuzzy shared secret data for secure key establishment. The method for device pairing proposed in the next chapters

[1]Note that for the best results the fingers are to be fixed, e.g. the distance between two fingers does not change while drawing the pictures. Moreover, the test have shown that drawing pictures with two different hands (at the same time, without lagging) does not yield in any usable results; the device will never pair since the two pictures are too different.

[2]Longer drawings usually have more recorded events.

[3]The Operating System (OS) of a specific device captures events in its own schedule; when the thread assigned for that gets its rights to run. Thus, the timing of the events and the duration between two consecutive recordings are not the same.

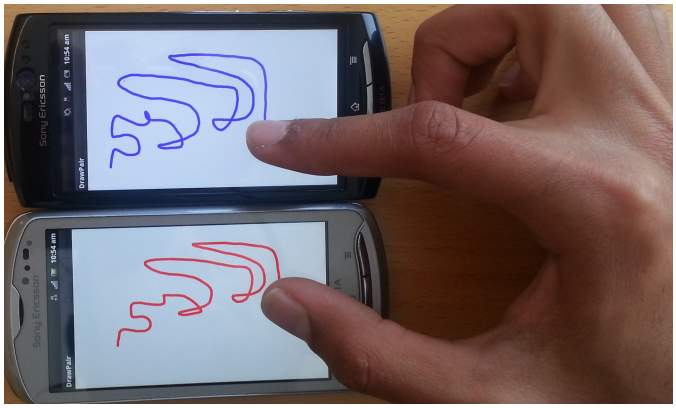


Figure 3.1: Synchronized drawing for device pairing

may be applied to smart phones, laptop touchpads, touch-sensitive control panels, drawing pads and certain types of mice. Pairing can occur between any two device types. However, due to the differences of the event recordings which are both device and vendor specific and due to the characteristics of recorded raw data the inputs must be processed to allow the use of a certain metric to compare the drawings.

This chapter focuses on finding a suitable metric for drawing comparisons and explains the collection of test data. The structure of the chapter is as follows. The first section explains the collection of the test data, devices used for collecting drawings and the user groups that participated in the research. The second section explains the implementation of the most forward metric based on Euclidean distance[4]. The third section includes the usage of Euclidean location metric along with its first derivative, time. The fourth section introduces *angle* metric and LURD metric as a good approximative metric for string encoding. The fifth section explains the results for all metrics and discusses viable solutions for the protocol.

3.1 User data collection

For the purpose of data collection we[5] have developed a smartphone, tablet and laptop touchpad applications that record finger strokes. These applications collect the user input and save it in the previously explained form. All subsequent encoding and calculations are performed off-line in Matlab.

[4]"In mathematics, the Euclidean distance or Euclidean metric is the "ordinary" distance between two points that one would measure with a ruler, and is given by the Pythagorean formula"[DD09]

[5]The original code for android phones was taken from Sethi et al.[SAA13]

Our user database consist of 24 test users. The users' field of occupation ranges from technical to non-technical users. To conduct a small user study, the participants were first told just to draw to synchronized pictures on two device that were chosen without any explanations or instructions given. This proved to be a **failure** since over 70% of the test drawings were too small pictures (quick drawing between 1-3 seconds) which are not enough for the protocol design[6]. Additionally, the users are not allowed to lift fingers after the drawing has started on both device. However, in more than 50% of the test drawings users tried to draw, for them, some known shape which included lifting fingers. These test drawings were considered incomplete and discarded accordingly.

Due to these disappointing results we decided to instruct users on how to use the application (and how they would use the real pairing mechanism when implemented[7]). The instructions were given orally and the process of users drawing was supervised to immediately eliminate sessions that do not fit the protocol design (more on these may be found in Chapter 5 about the security analysis of this device pairing protocol).

To conduct analysis on the appropriate metric we have gathered, on average, 15 test drawings per user. This number of individual test may give us a complete understanding about the similarities and difference among one user's drawings as well as the similarities and differences between drawings made by different users. In this way we can distinguish a metric that excerpts the biggest distance between *positives* (two matching drawings whose distance is less than some threshold) and *negatives* (distance is above threshold).

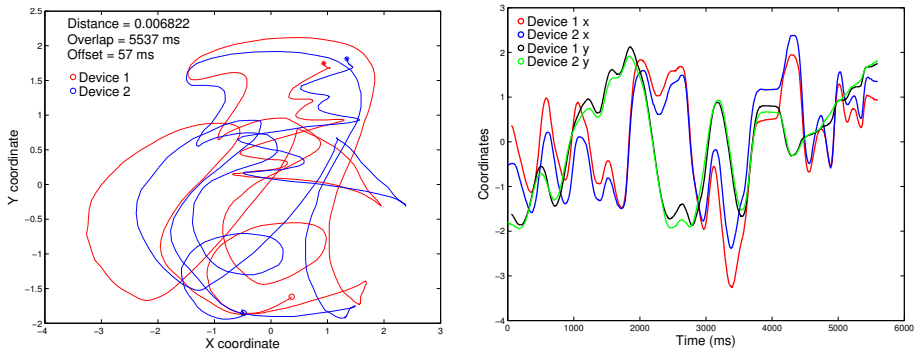
The scenarios used for analyzing different metrics are chosen randomly by taking 6-8 users with their measurements from the user database. In this way we overcome the possible lack of diversity when conducting tests since users have their own preferences and different device may have different specifications regarding touch sensitivity and screen resolution.

3.2 Location metric

In order to compare the drawings, we need to define a distance metric and a threshold value for accepting the drawings as the same (and in that sense end up with successful pairing). The same metric and threshold should work for all devices and users, so that the authentication can be used without a training period or setting any parameters. Our experiments (on different metrics) showed that such metrics and threshold can

[6]Shorter pictures imply less data which then limits the overall entropy. Thus, the security of the protocol is endangered.

[7]One possible way of giving instructions might in form of a pop-up banner that appears before the drawing process starts



(a) Centered and scaled drawings (b) x and y coordinates as function of time

Figure 3.2: Location metric for comparing the drawings

be found easily and that the comparison is not sensitive to minor changes in the parameters of the algorithms.

The first and most obvious metric that we use is based on location as a function of time. Figure 3.2a) shows two matching drawings recovered from their raw data using Matlab. Due to the fact that different touch surfaces differ in their dimensions, pixel density (resolution), aspect ratio and frequency of the sensed movements, we need to perform pre-processing to raw data in order to be able to use specific distance measuring functions. The set of needed actions is consisted of:

1. **Centering.** The drawings are centered so that the mean value of the x and y coordinates is zero. This is done by subtracting the mean value of x from every x coordinates, and for the y coordinate respectively,

$$x = x - mean(x),\ y = y - mean(y), \tag{3.1}$$

2. **Scaling.** Due to different sizes and pixel densities of touch surface scaling of the drawings is necessary so that the mean distance from the center along each axis is one unit. This is done by dividing each x coordinate with the absolute mean value of x set of coordinates, and as well doing the same for y coordinate respectively,

$$x = x/mean(abs(x)),\ y = y/mean(abs(y)), \tag{3.2}$$

3. **Interpolating.** Due to the different frequency of event recordings controlled by inner OS threads we needed to interpolate the location data to one-millisecond

frequency. We interpolate the data set (x and y coordinates) in respect to time (t as the third variable of the set),

$$\begin{aligned} x_i &= x_k + (t_i - t_k) * \frac{(x_{k+1} - x_k)}{(t_{k+1} - t_k)}, \\ y_i &= y_k + (t_i - t_k) * \frac{(y_{k+1} - y_k)}{(t_{k+1} - t_k)}; \end{aligned} \tag{3.3}$$

We also experimented with rotating the drawings, in case the two drawings are at an angle, but that proved unnecessary in practice, except for 90 ° and 180 ° turns. The 90 ° turn was explicitly needed in cases of using laptop touchpads since their aspect ratio is opposite of a tablet or phone aspect ration in natural standing position. The 180 ° turn might have been needed if the two phones or tablets were rotated on different sides but then the status bar of the apps would be opposite (hence, it is easily observable and natural to fix it immediately before starting to draw).

Figure 3.2b) represents the same drawings as function of time. We may observe one more step of processing the original raw data here. The graphs have been aligned on the time axis to minimize their distance by the chosen metric, which is very close to the time shift that also gives the highest cross-correlation. Basically, the value of the maximum offset is taken to be 500 ms and the distance between two drawings is calculated in a +- 500 ms frame to choose the value of the offset which minimizes the distance. In this particular case, the offset between the drawings is 57 ms from the start of the drawings, which is quite a typical value. Figure 3.2a) also indicates the start and end of the overlapping time period, over 5 seconds in this case.

The location-based distance metric is computed from the *centered*, *scaled* and *interpolated* data as the Euclidean norm of the difference between the x and y coordinate vectors using this formula:

$$\begin{aligned} &D(\text{drawing}_A, \text{drawing}_B) = \\ &\quad \frac{1}{n}\Big(\sum_{i=1}^{n}((x_{A,i} - x_{B,i})^2 + (y_{A,i} - y_{B,i})^2)\Big)^{1/2}. \end{aligned} \tag{3.4}$$

Above, the norm is is divided by the length of the drawing to avoid bias towards shorter drawings. The length of the drawings was taken into account separately by setting a minimum drawing length (4 seconds) and a maximum difference for the lengths of the drawings (500 ms).

3.3 Movement metric

Another idea worth mentioning is the expansion of location metric into movement metric. Basically, the distance metric could also be based on the movement, i.e. speed and direction of drawing as a function of time. In some applications, such as handwriting recognition, good results have been achieved by comparing the direction of strokes. We experimented with metrics based on the first and second derivatives of the location, i.e. velocity and acceleration. However, from the results we were not able to derive useful metrics. The positives and negatives of all measurement are all meshed up together without a clear gap that separates them. Therefore, it is impossible to set a threshold that has the characteristic of 0% false positives with a reasonably small percentage of false negatives that could be allowed(see Figure 3.3 under movement metric). The main reason is that the event-based API touchscreen APIs on the smart phones (e.g. *MOTION_EVENT* on Android and *touchesMoved* on iOS) do not give frequent enough readings for accurate calculation of the speed or acceleration of the finger.

3.4 String distance metrics

This and the next several subsections entail a different approach for calculating the distances between two drawings. Namely, the idea is to find a way to encode raw data into a form (i.e. string) that can be used for subsequent comparison and/or

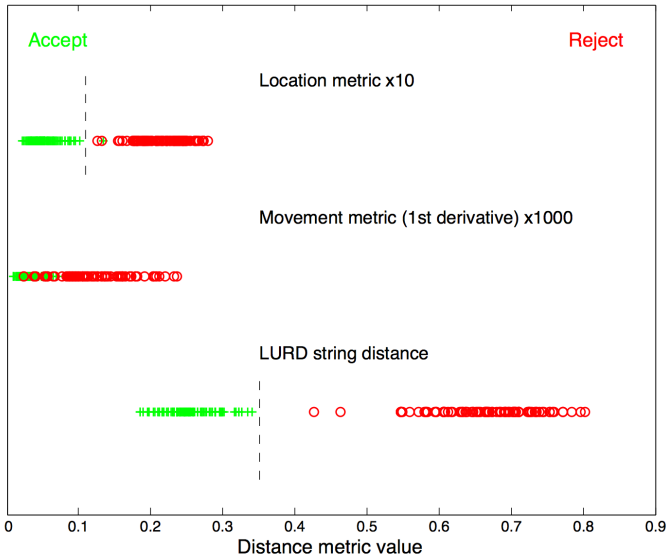


Figure 3.3: Comparison of distance metrics for drawings [SAA13]

for fuzzy extraction of the key. These forms use specific alphabets of symbols that a particular metric needs. After the two strings are created (each string represents an encoded drawing from one device) the distance between them is calculated using some well known function. However, most of the distance functions work with binary symbols (e.g. Hamming distance[8]) and have several constraints such as:

- mostly based on substitution of symbols,
- insertion and deletions of symbols are not considered,
- the length of both strings that are being compared has to be the same,
- a shift of only one character can have a major effect on results.

3.4.1 Levenshtein distance

By definition in the field of information theory, the Levenshtein distance (also known as *edit* distance) is a string comparison metric for measuring the difference between two strings. Particularly, a distance between two strings equals the smallest number of substitutions, insertions and deletions needed to change one string to another [Bla08]. The idea of edit distance originates from 1966. by V. Levenshtein where he had the goal of finding binary codes capable of correcting insertions, deletions and reversals (substitutions) [Lev66][Lev65].

In this thesis we firstly used the original Levenshtein distance to compare our matching and non-matching strings which yielded in fairly good result. By closely observing the implications of our metrics (except the location metric) we came to a conclusion that a specific sub-version of Levenshtein distance, called Weighted Levenshtein distance, applies better. Specifically, the difference cost between two symbols should not be the same for all combinations since it is not the same error extent if the original drawing deflects for $10\,^\circ$ rather than, for example, $50\,^\circ$. Therefore, here we will examine some properties and implication of Weighted Levenshtein's distance to furthermore implement and apply it to our metrics.

According to Tanaka et al. [TK76], let A and B be two finite arbitrary size sequences of symbols from a specified alphabet. If sequence A can be transformed to B by applying insertion to n symbols, deletion to d symbols and substitution to s symbols then the weighted distance might be defined as:

$$W_{LD}(A \to B) = min_i(pn_i + qd_i + rs_i), \tag{3.5}$$

[8]In information theory, "the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. In another way, it measures the minimum number of substitutions required to change one string into the other, or the minimum number of errors that could have transformed one string into the other."[Ham50]

where p, q, and r are positive floating points representing weights for insertion, deletion and substitution respectively. However, if $W_{LD}(A \rightarrow B)$ is written in the form as above, then $W_{LD}(A \leftarrow B)$ has a slightly different form defined as:

$$W_{LD}(A \leftarrow B) = min_i(pd_i + qn_i + rs_i). \tag{3.6}$$

From these two equations it follows that if $p \neq q$ then the values of $W_{LD}(A \rightarrow B)$ and $W_{LD}(A \leftarrow B)$ may not always be equal. Yet, the metric itself holds some usual properties:

1. If p and q equal ∞, $r = 1$, and $\mid A \mid = \mid B \mid$ where $\mid A \mid$ means the length of the sequence A, then $W_{LD}(A \leftarrow B)$ is the Hamming distance between two strings,
2. If $W_{LD}(A \leftarrow B) = W_{LD}(A \rightarrow B)$, then $p = q$. The weighted metric is then called true metric, and it can be specified as:

$$W_{LD}(A, B) = min_i(p(d_i + n_i) + rs_i), \ and \tag{3.7}$$

3. If $p = q = r = 1$ then the weighted Lensnhtein metric is obviously equal to the original Levenhstein distance.

The difference that characterizes the Levenshtein distance from others (i.e. in this case Hamming distance) is that the space of sequence includes all lengths incuding zero-length sequence and is defined as $S(= \cup_i S_i)$ where as the space of the Hamming distance is defined in a fixed length space sequence S_k, where k denotes its length.

The computation of the Levenhstein distance ($W_{LD}(A, B)$) can be performed in several ways, yet the two most popular are via recursions or the matrix approach. Due to the preferences and setup of our testbed solutions we currently use the matrix approach (quick computations in Matlab). The method was originally proposed by Sankof [San72], while the closest algorithm for calculation that we use is of Wagner et al. [WF74]. Computation of the edit distance is therefore based on the fact that if we can create a matrix that can contain edit distances between all prefixes of the first and second string (respectively), then we can use techniques of applied (bottom-up) dynamic programming [BD62] to compute the matrix values. Hence, the last value that is computed (two full sized strings) represent the distance between our two sequences. A mathematical representation of the method is as follows [TK76]:

– Let $A = x_{11}x_{12}\cdots x_{1k}$ and $B = x_{21}x_{22}\cdots x_{2l}$, where x_{ij} is a symbol. We can construct a recursive relation as:

$$\begin{aligned} W_{LDij} = \min(&W_{LD_{i-1,j-1}} + r_{i-1,j-1}, W_{LD_{i-1,j}} + p_{i-1,j}, \\ &W_{LD_{i,j-1}} + q_{i,j-1}), \ if \ i,j \geq 2, \ and \ p == q, \end{aligned} \tag{3.8}$$

where $r_{ij} = 0$ when $x_{1j} = x_{2i}$ and $r_{ij} = r$ when $x_{1j} \neq x_{2i}$. The coefficients for $p == q$ depend on the actual values of (x_{1j}, x_{2i}) and are calculated via separate function as:

$$\begin{aligned} N &= M/2, \\ cost &= N - abs(N - abs(symbol1 - symbol2)), \\ cost &= 2/N * cost, \end{aligned} \tag{3.9}$$

where the M is the size of the alphabet and we suppose that the symbols are pure integers (so they can be subtracted accordingly). The end value of the distance $W_{LD_{k+1,l+1}}$ is furthermore calculated under these conditions:

1. $i \leq k + 1$, $j \leq l + 1$,
2. $W_{LD_{1,1}} = 0$,
3. $W_{LD_{i,1}} = (i - 1) * p$,
4. $W_{LD_{1,j}} = (j - 1) * q$.

Thus, the final value $W_{LD}(A, B) = W_{LD_{k+1,l+1}}$.

3.4.2 ANGLE metric

The idea behind the ANGLE metric is to try to encode the original drawing to be as precise as possible, yet enough robust so the distances between strings of two drawings are reasonable for matching drawings and large for non-matching drawings. Both ideas are based on the fact that the encoding starts from the first recorded point (x and y coordinate) and then we move forward a certain distance d in the direction (characterized by angle) of the second point until we reach it. Upon reaching the second point, the encoding continues in the direction of the next point and so forth. There were two initial ideas how to encode those angles and distances:

1. The metric is based on two variables δ and α. Variable δ represents the distance traveled in pixels while α is the angle that is characterized by the number of degrees that the encoded line can travel either to the left or to the right of the original drawing (i.e. line between two consecutive points).Namely, the alphabet of the metric is simply binary where 0 represents the movement to the left and 1 represent movement to the right of the original line. Every movement for a distance δ is then recorded with one of the symbols, ending the string upon reaching the last point of the original drawing.
2. The second idea also consist an angle and distance d, yet is much different. In contrast to the binary alphabet of the first angle metric, here we have an alphabet of $0...M$ symbols that depends on the number of possible angles in which the encoded line can travel. Distance d is the minimum number of pixels to move from one point to another. Moreover, the way of encoding the string

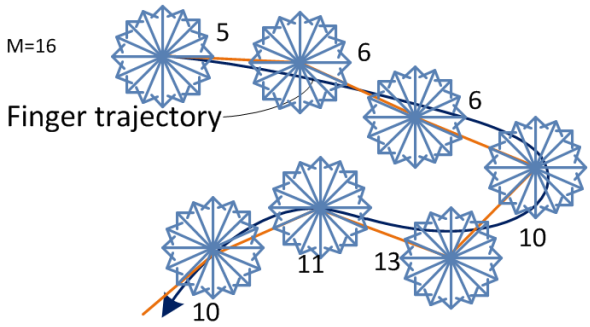


Figure 3.4: ANGLE string encoding

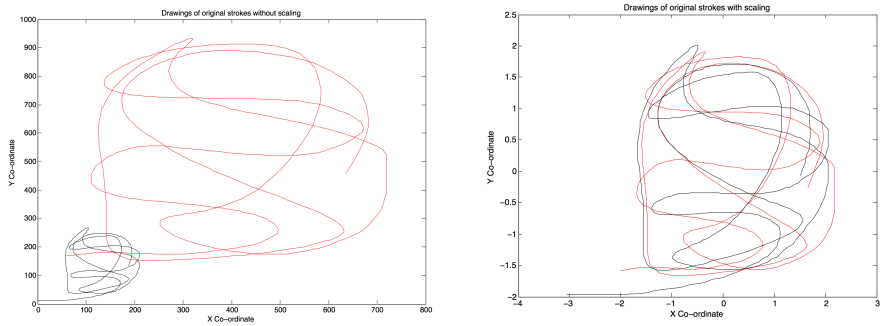
for comparison includes not only the angle but the number of distances traveled in each direction. Figure 3.4) shows the idea for encoding the line as described above.

The first metric design was tested with different angles and distances and did not yield very good results. The threshold for accepting the drawings as matching could not be set in a way that the rate of false positives is 0 while keeping a reasonably small rate of false negatives. Therefore we have decided to drop the metric and focus more on the second idea.

However, initial tests of the second idea have proved to be good. Therefore, we continued to develop the second idea and find the most appropriate solution for our device pairing protocol.

The encoding algorithm is divided into several steps. The first important step is to *center* and *scale* the drawings in the same way that it was done for the *location metric*. In this way we eliminate the problem of different touch surface sizes and their resolutions. Figure 3.5 shows how the drawings with and without scaling and centering look like. It is easily observable that even though the pictures look extremely similar it is impossible to compare them (i.e. encode them in a way that they are comparable). The second step is to find the mean values of all 4 vectors that represent x and y coordinates of both drawings. **The final value that we seek is the *minimum* of those calculations called *ratio*. The purpose of it is to divided the pixel distance variable d for movement between points. In this way the encoding of each angle is scaled properly as well.**

The last step of the encoding process is shown by Algorithm 3.1. The procedure takes x and y vector coordinates, pixel distance, number of maximum angles and ratio as arguments. To be able to calculate angles via arctan function we need the length of all triangle cathetus between adjacent points. Hence, we calculate those



(a) Original drawing without modification (b) Centered and scaled drawings

Figure 3.5: Scaling properties for the ANGLE metric

in vector form as $dx\ dy$ for the x and y coordinates separately in respective order. These values are then used for each step to calculate the angle of the original drawing between two points. Afterwards, that angle is rounded to the closest of the available angles allowed by the metric alphabet[9] (10th row). Now we calculate how many steps of length d (scaled) we may move between those two adjacent point (11th row). However, the calculated number of steps is in the real direction of the original line and not in the direction of the rounded up angle so we need to calculate the difference errors from the point given by moving towards the encoded angle and the point of the original drawing (12th and 13th row). This is taken into concern when calculating the next angle, and each error is then fixed propagating the value to the last point (14th and 15th row). The calculated angle and the number of distance steps are then saved into two variables (16th and 17th row).

The last part of the Algorithm actually forms the string from the calculated values that is going to be used for drawing comparison (20th-27th row). For each angle stored as $angles(i)$ we fill the variable str with that exact angle for $distances(i)$ times. For example, if $angles = [10, 22, 15, 43]$ and $distances = [1, 2, 0, 2]$, the string for comparison will result in $str = [10, 22, 22, 43, 43]$.

The actual value of variables M and d are discussed in the results section of this chapter where we compare results of different metrics, test variations of good ones, and choose variables based on the values of False positive rates (FPR) and False negative rates (FNR). Note that the solution for the metric described above has a good mathematical standpoint and is fairly understandable. Yet, due to large number of calculations the method is rather computationally expensive. In the next several

[9]For example, if $M = 64$ the minimum angle step is $\alpha = 360\,°/64 = 5,625\,°$. So, possible angles are $0...M * 5,625\,°$

Algorithm 3.1 Encoding process for ANGLE metric

```
1: angles = []                                          ▷ Instantiating default variables
2: distances = []
3:
4: procedure encodeAngle(x, y, d, M. ratio)
5:     D ← d/ratio
6:     dx ← x(2 : end) − x(1 : end − 1)
7:     dy ← y(2 : end) − y(1 : end − 1)
8:     for i ← 1 : (length(x) − 2) do
                                                        ▷ angle encoded as 0..M-1
9:         degree ← arctan(dy(i), dx(i))/(2 ∗ pi)
10:        angle ←  mod (round(M ∗ degree), M)
                                                        ▷ distance in steps of length D
11:        distSteps ← round(√(dx(i) ∗ dx(i) + dy(i) ∗ dy(i))/D)
                                    ▷ Take the rounding error into account in the next step
12:        errX ← dist ∗ D ∗ cos(2 ∗ pi ∗ angle/M) − dx(i)
13:        errY ← dist ∗ D ∗ sin(2 ∗ pi ∗ angle/M) − dy(i)
14:        dx(i + 1) ← dx(i + 1) − errX
15:        dy(i + 1) ← dy(i + 1) − errY
                    ▷ Save the calculated angle and numberOfMovements for step i
16:        angles(i) ← angle
17:        distances(i) ← distSteps
18:    end for
19:
                ▷ Combine results from angles and distances into an encoded string
20:    k ← 0
21:    str = []
22:    for i ← 1 : length(angles) do
23:        for j ← 0 : distances(i) do
24:            str(k) ← angles(i)
25:            k ← k + 1
26:        end for
27:    end for
28:
29:    return str
30: end procedure
```

subsections we present an approximate metric which requires much less computation.

3.4.3 LURD metric

While the location-based metric and the angle metric are otherwise excellent for our purposes, they are fairly expensive to compute on a mobile device. The main cost in location-based metric is to find the exact time offset that best aligns the two drawings and in the angle metric it is the calculation of the encoding drawing. For this reason, we wanted a distance metric for the drawings that takes little computing power, does not require accurate time synchronization, computation of the exact time offset between the drawings or numerous angle calculations, and nevertheless accurately compares noisy drawings. Shape recognition algorithms that are based on approximate string matching are known to have these properties [KB02][Mae91]. Like the *angle* metric that we proposed, these algorithms encode the drawings into a string, which can be then compared using approximate string comparison algorithms (here we will also use Levenshtein's distance).

The approximate metric suggested for the pairing protocol is called LURD (originates from Sethi et al. [SAA13]) string distance. The basic idea is to record the movement of the finger in four different directions and encode it into string of letters where: L=left, U=up, R=right, D=down, and then compute a string distance between the two drawings.

The algorithm for encoding the drawing into a string is done as follows. Firstly, we have a parameter that is subject to change, more precisely the metric needs to be analyzed in order for us to decide which value is to be set. Namely, we are talking about dividing the covered area of the touch surface into a grid of $G \times G$ squares. Each movement of the finger that crosses at least one grid line is mapped to a string that has one character for each crossed grid line. The characters indicate the direction of the crossings. The first step is to interpolate the vectors (x and y coordinates) in a similar way done with the location metric, yet here we interpolate based on the grid size G:

$$\begin{aligned} x1min &= min(x1), \\ x1max &= max(x1), \\ x1 &= floor(((x1 - x1min)/(x1max - x1min)) * G), \end{aligned} \tag{3.10}$$

After that, two temporary strings are created that incorporate distances between adjacent coordinate points of the original drawing. These strings are used in a later stage for comparing their adjacent values and based on these we form the final string with letter l, u, r, d (e.g. $if\ tempX(i) \leq tempX(i+1)\ then\ str_i \leftarrow=' r'$).

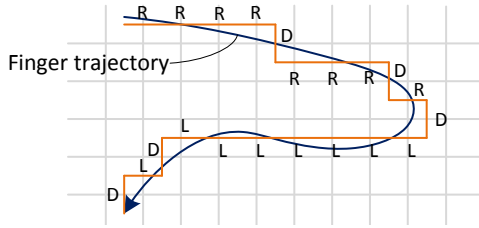


Figure 3.6: LURD-string encoding [SAA13]

For example, if the movement crosses one vertical grid line towards right and then one horizontal grid line in the downward direction, the string will be "RD" (see Figure 3.6 that illustrates the encoding). In the very rare cases where the crossing happens exactly at (or within the rounding error from) the junction of two grid lines, the crossing of the vertical line is recorded first. The substrings from the recorded movement events are concatenated to represent each drawing. The two drawings are then compared using a weighted string distance metric (Levenshtein distance). The function is specific for this metric and gives weight 1 to each deletion or insertion of characters, and for the substitution we have weight of 1 for 90 ° angle turns (e.g. d to l or r, r to u or d) and weigh of 2 to for 180 ° turns (e.g. u to d, l to r). However, the actual encoding and the comparison of the strings is not done using the LURD alphabet. We have come to a conclusion that the computations operate more faster with numbers rather than ASCII characters (due to constant conversions). Hence, the letters from L-U-R-D are transferred to number 0-1-2-3 respectively.

As we will see in the discussion (next section), this approximate string distance metric satisfactorily separates the synchronized drawings from pairs of unrelated drawings. Moreover, it automatically synchronizes the two drawings by inserting characters to the beginning of the one that starts later. There are simple and efficient algorithms for computing such string distances [YB07], and these proved to work well on the mobile devices.

3.4.4 LURD xy metric

LURDxy metric is a sub-version of the LURD metric intended to improve the outcome of the string distance comparison. The idea is to overcome and reduce the number of insertion, deletions and substitutions by splitting the complete movement into two separate string. Particularly, the movements are viewed as separates strokes made on $x - axis$ and $y - axis$. The algorithm of encoding is the same as in the LURD string distance except the last part where the string for comparison is created. After interpolating the coordinates into the grid we form two additional temporary substrings. The first one contains movement only on the $x - axis$ (L-R) and the

second one movements on $y - axis$ (U-D). Lastly, these strings are concatenated to form a final string for calculating the distance between drawings. Hence, the total amount of symbols (and number per each symbol) are the same but the order is rather different. The idea came to life by analyzing the structure of LURD strings where we have noticed that long movements in one direction can be interrupted (shaky hands) by a slight movement in an 90 ° angle and this may be affected by the precision of the touch surface. Therefore, by separating opposite-axis movements we measure strokes on each axis separately and avoid useless *errors* in the string. However, as we may see, the results, presented in the discussion chapter, did not bring reasonably well improvement.

3.4.5 LURD binary metric

LURD binary string metric is another sub-version of the original LURD metric. The need for this metric stems from the characteristics of error correction codes. Specifically, our design of a fuzzy device pairing protocol comprises the usage of error correction codes to successfully derive a shared secret between two different devices. The characteristics of the protocol, its sub-parts and the mathematical background that supports its creation are explained in detail through the next several chapters. However, preparing the raw data and encoding it to be correspondent to the protocol design is of utmost importance. Most error correction codes and formulations work on binary code words. Therefore we need to find a metric that can either be originally in that form or be transformed to it.

The LURD binary string metric does not differ from the original LURD metric in its algorithm for encoding, yet only the last part including the formation of the comparison string is changed. Particularly, the lastly formed LURD string is transformed in binary form by encoding each symbol of the LURD metric alphabet: $L - 00$, $U - 01$, $R - 11$, and $D - 10$. The drawback of this encoding is that the strings used for comparison are now twice the size of the original LURD but since we are only working with the binary alphabet, the Levenshtein distance function operates much faster. Another drawback that we encountered is a slight impairment of the overall results, where the LURD binary metric has a shorter gap distance between matching and non-matching results. However, this does not effect the feasibility of the protocol since the test results show (see the discussion section) that it brings more benefit when used in the fuzzy device pairing protocol.

3.5 Metric evaluation and analysis

In this section we will first evaluate the metrics separately with the goal of finding parameters for them which give the best results. After that the metrics are compared all together to choose the best suitable metric for our device pairing protocol. The

evaluation of each metric is based on fitting the test results to a normal distribution and then calculating false positives rates (FPR) when given the value of the false negative rate (FNR). Formula of the Normal distribution that is used is:

$$f \sim \mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.11}$$

The idea is to first find a threshold on the x-axis using the Inverse cumulative distribution function (ICDF) with the input of $1 - FNR$, mean value and standard deviation of test data that is supposed to be accepted by the protocol. That threshold is then used as an input variable to the original Cumulative distribution function (CDF), along with the mean value and standard deviation of the test data that is supposed to be rejected by the protocol, in order to calculate the false positive rate. Formula used for the CDF is as follows:

$$f \sim \mathcal{CDF}_{\mathcal{N}}(x, \mu, \sigma^2) = \frac{1}{2} * (1 + erf(\frac{x-\mu}{\sqrt{2\sigma^2}})) \tag{3.12}$$

Obviously, we want to minimize the FPR since if any of the false positive occurs, the protocol is not sustainable. On the other hand, we want to minimize the FNR as well since the protocol has to be user friendly with minimum of false negatives (users could get discouraged if many of the pairing attempts results in a failure). For many analysis purposes, we took that the FNR is equal to 1% and then tested the value of FPRs while changing parameters of the metrics.

3.5.1 Optimal angle number and distance for the ANGLE metric

As explained in the previous sections, the ANGLE metric is characterized by two variables: the number of angles M and the distance d. First tests of the metric were performed with the distance $d = 20$ and $M = 32$. **Keep in mind that the distance d is presented here as a number of pixels while in the actual implementation it is scaled in compliance to the scaled drawings. Hence, these number can be used to compare the correctness of the metric since they linearly suit the used ones.** Our observation and initial conclusion was that the encoding will be more similar to the original drawing if we have more angle values that we can move our points to. The decision was to double the number of angles to $M = 64$. So we ran a test that checks the value of FPRs while the FNR changes gradually by 1%. Results are shown in Figure 3.7 where we may observe that two lines characterizing 32 and 64 as the number of angles follow each other perfectly. Reasons for that lie in the use of modified weighted Levenshtein distance. Even though the original drawing may not be followed perfectly, due to the small

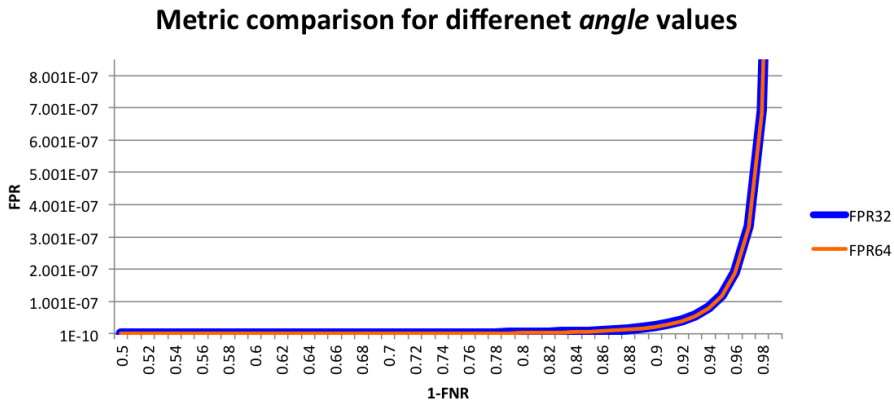


Figure 3.7: Comparison of *FPRs* in regards to different angle values and FNRs

distance $d = 20$ and the weighted distance calculation, the results do not change at all. Additionally, it is worth mentioning that the computation costs also do not increase even though the alphabet has more symbols (when $M = 64$). In conclusion, related to the angle value, we have decided that the value of 32 angles stays as a fixed parameter in further tests regarding distance d.

Next step is to find the most suitable parameter d for the metric. To test that we form a test scenario where the weighted Levenshtein distance is calculated for all encodings, ranging from $d = 10$ up to $d = 200$ with the step of only 2 pixels (after the value of $d = 60$ we increased the step to 10). During those tests we calculate the FPR based on the value of FNR which is equal to only 1%. First test incorporates the distance d without performing centering and scaling operations. Graph shown on Figure 3.8 represents the change of the FPR and Threshold of acceptable drawings on x-axis in regards to the changing (non scaled) distance d. Line on the graph show chaotic behavior which can easily be explained by the fact that the distance is not scaled properly. Namely, various different device have a wide range of resolutions and sizes of the touch surface. Taking that into consideration along with the fact that different devices do not call the touch sensing events in their OS on the same time basis we end up with unpredictable results that may vary (for example, if we add or remove a certain device the results change drastically).

However, the most important test for our metric is when the distance d is scaled. Figure 3.9 represents a graph with same properties as in the previous case but with the fact that the distance is scaled. Our goal is to find a maximum value of the distance that at the same time has an acceptable level of FPR. Maximization of the distance arises from the fact that the computation cost of weighted Levenshtein

distance is increasing with lower distance since low distance values imply more artificially created points for encoding and thus longer strings to compare. Having a look at the Figure 3.9 and by observing statistical data we come up with distance $d = 34$. The actual minimum is at the distance of 36 but the value of FPRs are different on a scale of 10^{-10} and then it is more useful to reduce computation costs.

It is also interesting to see how the ANGLE metric performs in regards of encoding the original drawings. Figure 3.10 shows 6 different subfigures, each containing 4 lines. Two lines (red and black) show the original drawings made by the user (accepted pairing instance of similar drawings). The other two lines (blue and green,

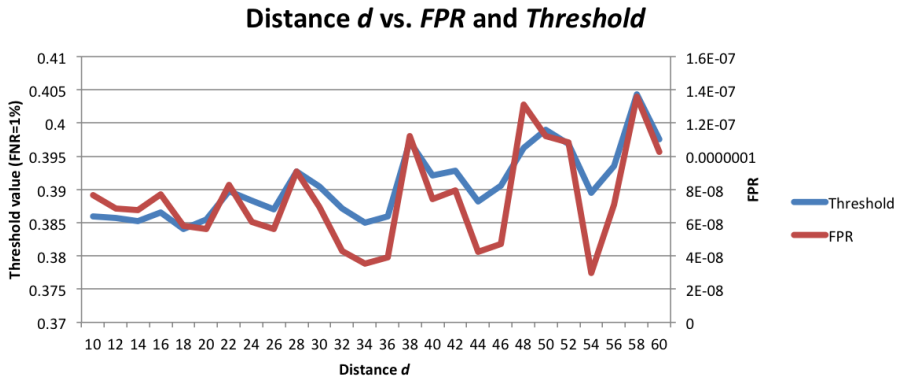


Figure 3.8: Comparison of *FPRs* and *Thresholds* in regards to changing distance d (NOT scaled)

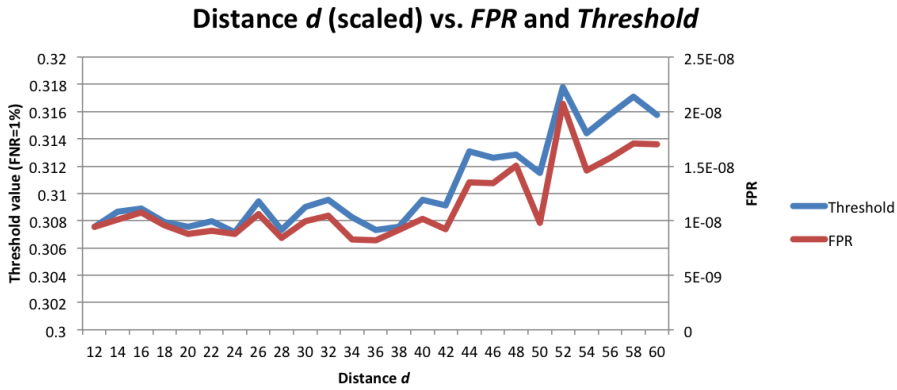


Figure 3.9: Comparison of *FPRs* and *Thresholds* in regards to changing distance d (scaled)

respectively) show drawings recovered from the calculated movements of the angle metric. We may observe a gradual change when increasing the distance d where the encoded drawings of smaller distances follow the original line almost perfectly. If we increase the distance up to 200 the line cannot be followed precisely any more and the encoded drawing shows hard turns. In that case even small deviations in the original drawing will cause the ANGLE metric to encode movement points under different angles which affect the calculation of the Levenshtein distance greatly.

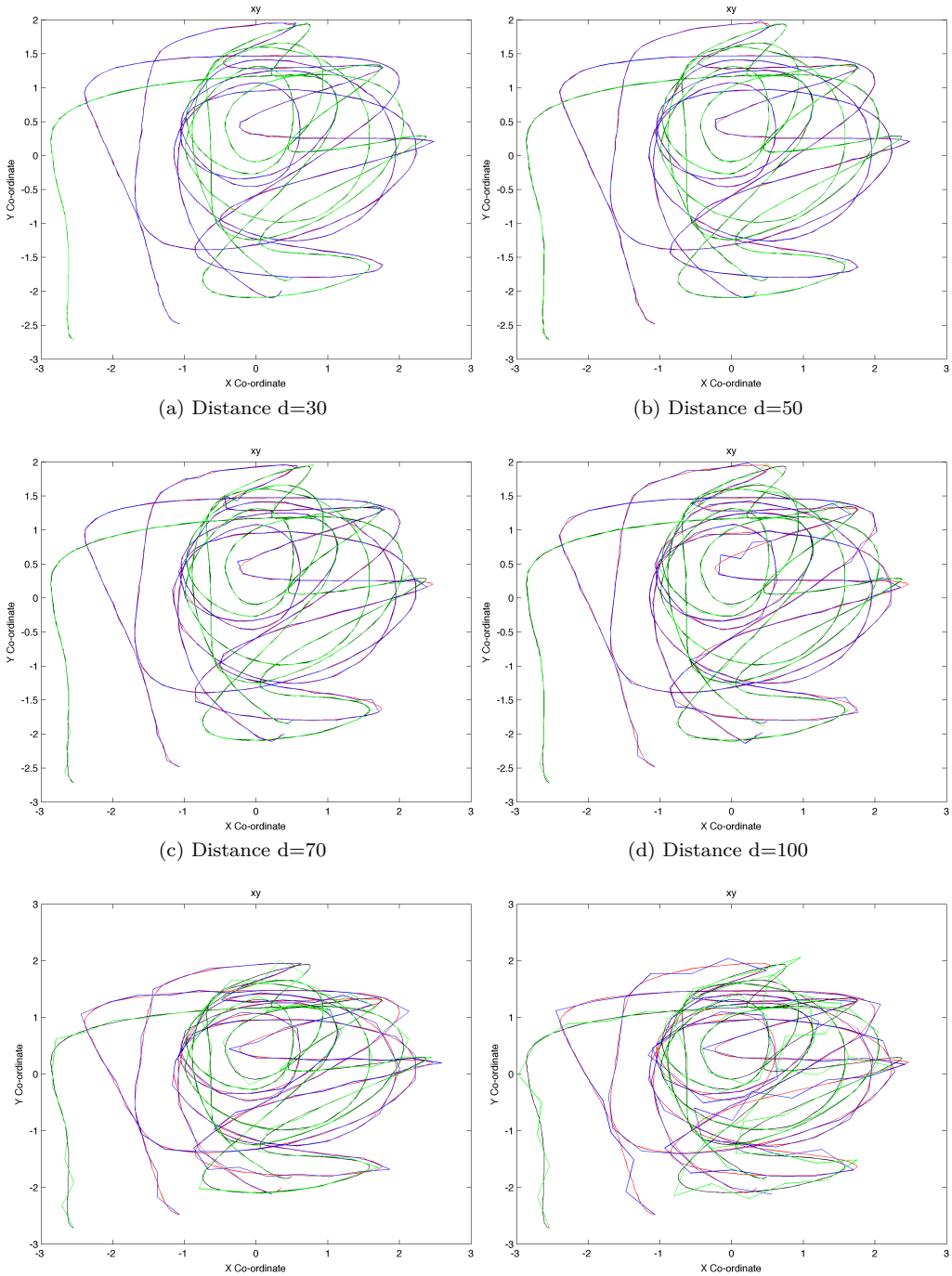
Another graphical representation of the results might be seen in Figure 3.11. Subfigures from *a)* to *f)* show distribution of the matching (green) and non-matching (red) test results. Each subfigure is characterized with a different value of distance d (the same values as in Figure 3.10 for easier comparison)[10].

From the figure we may notice that with the increase in distance the gap between the positives and negatives reduces, and eventually overlaps. Our goal is to maintain the biggest possible gap while taking into consideration the overall computation costs. Therefore, distance of 36 again proves to be appropriate since the gap is big enough to allow variations of results without endangering the security of the protocol.

3.5.2 Optimal grid size values for the LURD metric

Unlike the ANGLE metric, the LURD metric is characterized by only one parameter, the size of the grid G. The optimization of the grid size is carried out by running test scenarios on the user group, but each time with a different grid size. The grid size variation ranges from 2×2 up to 64×64 with the increase step of 2. We took the FNR to be 1% and tested the possible values of the FPRs and Thresholds on the x-axis. The result are shown in Figure 3.12. *xThresholds* represents the value of the threshold that is calculated via ICDF by using the mean and standard deviation values of matching drawings along with the mentioned FNR. Next, that values is used in the CDF to calculate the FPR. Note that the FPR is presented in the figure on a logarithmic scale due to the large differences between the values. Specifically, up to the level of 12×12 grid size the FPR value is so large that we could basically not see the variations for larger grid sizes. By observing the figure and the statistical results we conclude that the minimum FPR is given when the grid size is $G = 58$: $1.54256 * 10^{-7}$. Due to the large number of squares the encoded strings are quite long and the computation costs increase exponentially. Therefore, we search for a more suitable grid size but without losing the quality of the metric. One of the appropriate sizes is $G = 29$ with the value of FPR equaling $2.60326 * 10^{-7}$. The extent of the loss in the metric quality is not meaningful, yet the computation cost is much less since the grid size is half of the original one with the minimum FPR.

[10]Note that the test on the subfigures *d)* to *f)* were made on a smaller user group, so the bars of positive matches look larger even though the scale of the y-axis is changed.



(a) Distance d=30

(b) Distance d=50

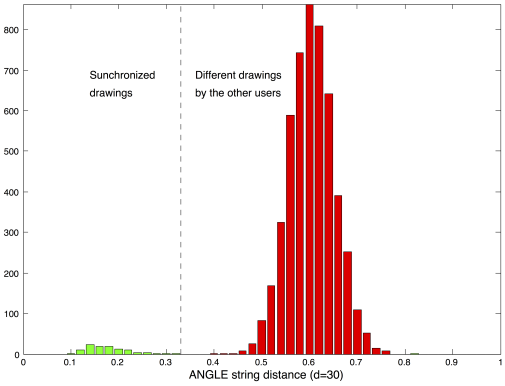
(c) Distance d=70

(d) Distance d=100

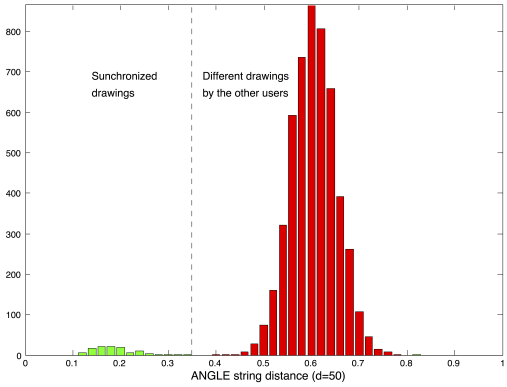
(e) Distance d=150

(f) Dist,ance d=200

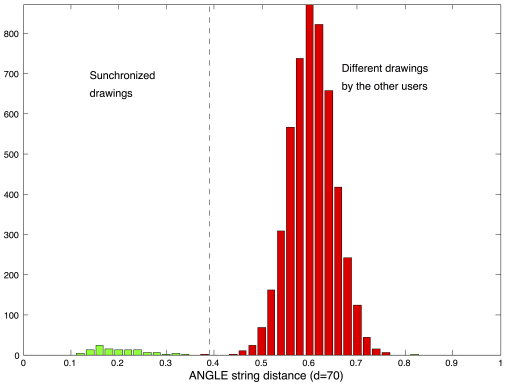
Figure 3.10: ANGLE metric original+encoded drawings over different distance values



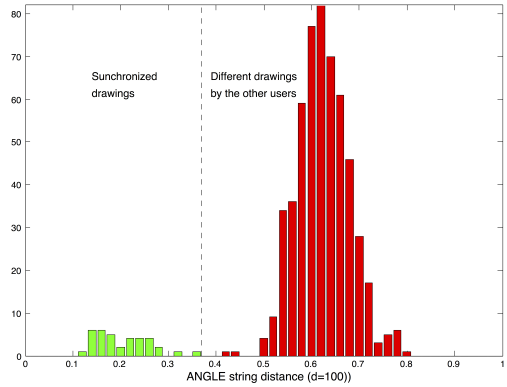
(a) Distance d=30



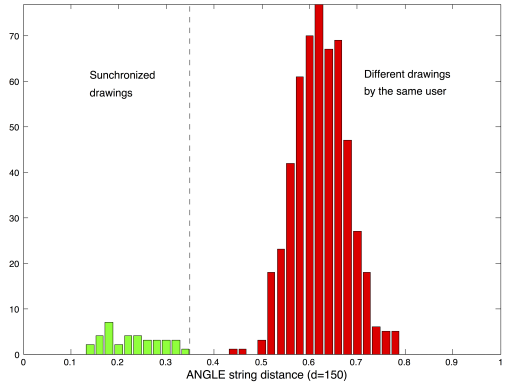
(b) Distance d=50



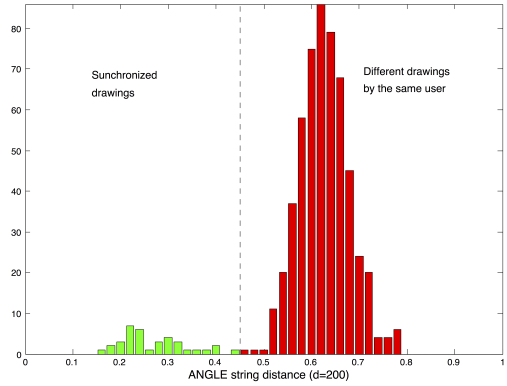
(c) Distance d=70



(d) Distance d=100



(e) Distance d=150



(f) Distance d=200

Figure 3.11: ANGLE metric distribution over different distance values

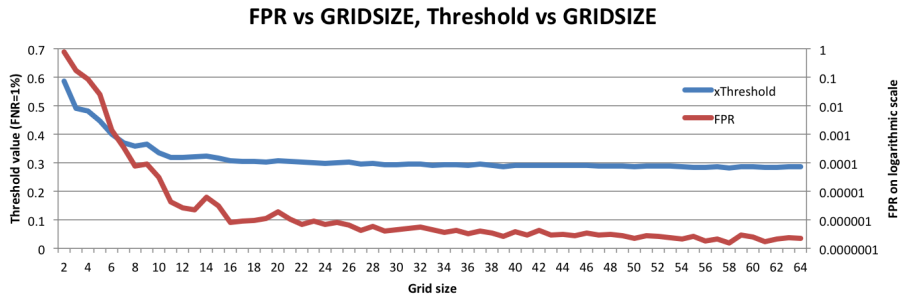


Figure 3.12: Comparison of *FPRs* in regards to the number of angles M

3.5.3 Comparison of metrics

The final step of the analysis is comparing proposed metrics that have optimal parameters. Thus, we take the ANGLE metric with $M = 32$ and $d = 36$ and LURD metric (along with its subversions) with $G = 29$. Figure 3.14 shows the distribution charts for each of the compared metric. From here we might observe that the ANGLE metric (Figure 3.14c) shows the best results, keeping in mind the size of the gap between the matching and non-matching results. Next is the LURD metric (Figure 3.14d) with slightly worse results. However, it might be difficult to distinguish the quality of the metric only from the distribution charts. Hence, in Table 3.1 we show the numerical representation of the results, and also the graphical representation of FPRs (on a logarithmic scale because of greater variations) by each metric with the changing value of FNR (Figure 3.13). The figure clearly shows the difference between metric qualities. The best performing is the ANGLE metric with the lowest value of FPR when FNR is 1%. It is followed by LURD metric which shows lower quality results on a scale of 10^-7. Next is the location metric with worse results approximately on the scale of 10^-3. LURD binary follows and interestingly shows worse results in comparison to the regular LURD metric on a scale of 10^-5. Yet, by looking the distribution of the LURD binary we can see a clear gap between the matching and non-matching results. Thus, the higher value of the FPR might be explained by a slight deviation from the Normal distribution (we may see that the first bar in the Figure 3.14f is large).

Taking these result into consideration we may conclude on which metric to use in the protocol. Even though the ANGLE metric shows the best results, the calculation process has a high computation cost due to the high number of movement points and calculations of suitable angles and error propagation. Hence, LURD metric seems as a good approximation of the more mathematically stable solution (angles and movements). LURD metric in its description has a drawback that a lot of information is lost during the encoding, yet the performance results are good and do not show

Table 3.1: Comparison of metric results (FNR=1%)

Metric	xThreshold	FPR
LURDxy	0,346822795	1,58167E-01
Movement	0,07634962	1,45685233E-01
LURD binary	0,336924031	8,68675E-04
Location	0,097337955	1,57469E-05
LURD	0,3281627109	2,60326E-07
ANGLE	0,319844303	1,3403E-07

that this lost is affecting the protocol execution. In conclusion, the commitment protocol from Sethi et al. [SAA13] will continue to use the originally presented LURD metric while we consider the binary LURD metric in out fuzzy device pairing protocol. The reasons for that lie in the fact that our metric comparison functions (along with the codes and error correction capabilities) rely on binary symbols. Even though the metric has worse results than the original LURD metric, test have shown that it would be enough to support the pairing mechanism.

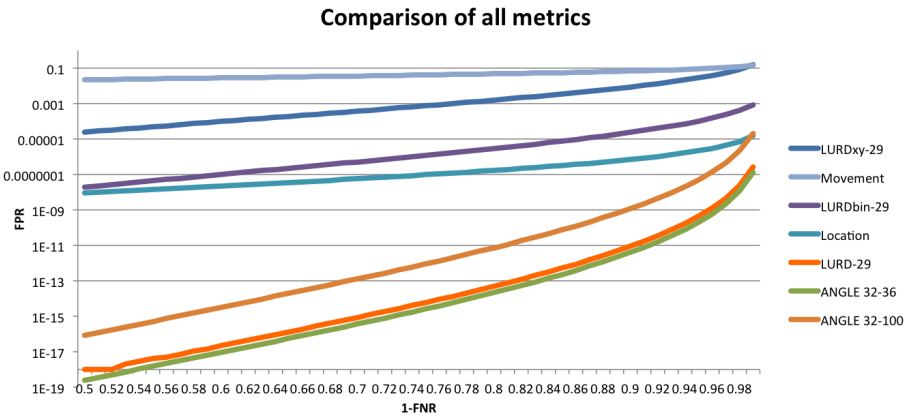
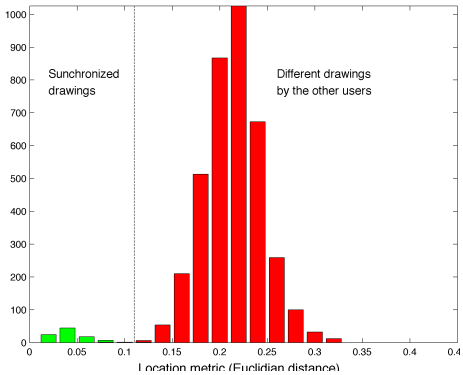
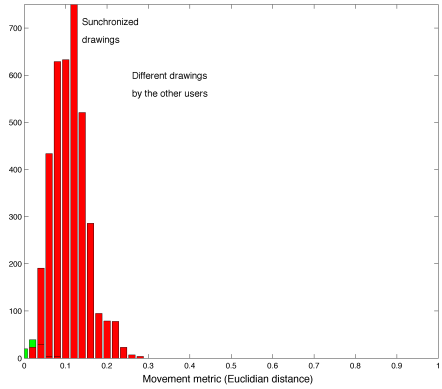


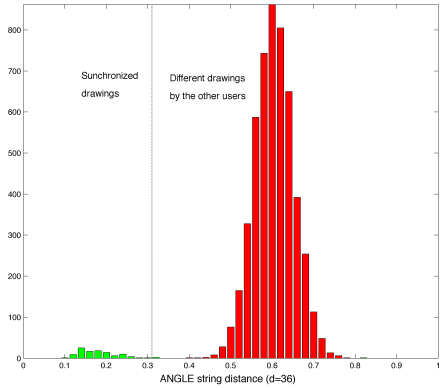
Figure 3.13: Comparison of *FPRs* achieved with LURD and ANGLE metric



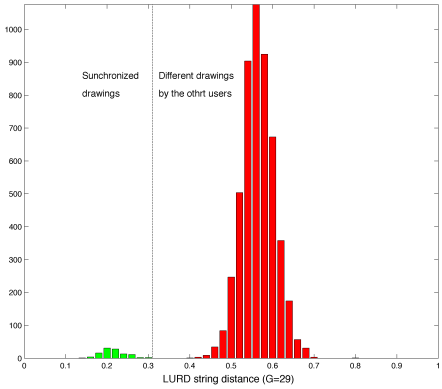
(a) Location metric



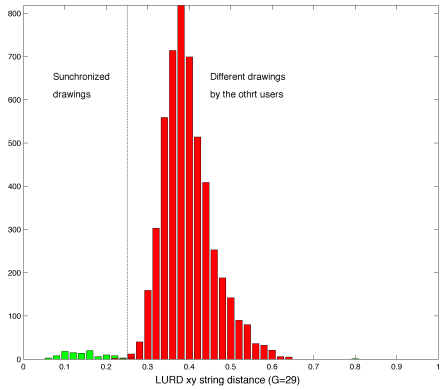
(b) Movement metric



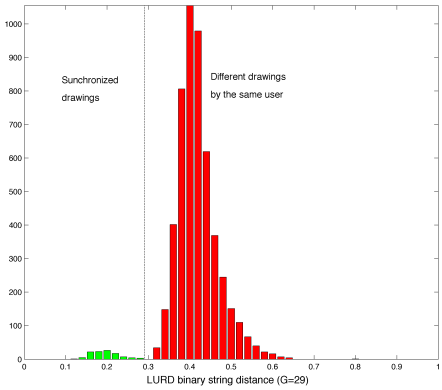
(c) ANGLE metric



(d) LURD metric



(e) LURD xy metric



(f) LURD binary metric

Figure 3.14: Comparison of all metric distribution graphs

Chapter 4

Introduction to fuzzy cryptography and component constructions

Cryptography in general is based on the quality and security of the secret that is used to protect the communication. Thus, uniformly distributed random strings, with the properties of precise retrieval, represent the best possible secrets. However, in practice it is often difficult to generate uniformly distributed secrets and at the same time be able to reproduce the same strings. Our protocol suffers from that exact problem since it is based on extraction of a shared secret from a fuzzy environment. Two drawings, each drawn on a separate device, may never be the same, yet just sufficiently similar. In this section we focus on the mathematical background needed for using such unreliable and non-uniform secrets to perform protocol bootstrapping and thus assuring secure communication.

Our main problem and therefore goal is to convert noisy non-uniform inputs (drawings) into reproducible, uniform and reliable random strings. A primitive that provides a solution was first introduced by Dodis et al. [DRS04] under the term *fuzzy extractors*. A fuzzy extractor extracts a uniformly distributed random string x from the noisy input ω, but with a certain degree of noise-tolerance. Noise-tolerance implies that the random string x can be reproduced exactly if the individual input ω changes to some ω', but remains close enough. However, the main question is how can the fuzzy extractor output the same random string even with the change of the input. In order to do so, a fuzzy extractor outputs another **non-secret** string, named P, which is subsequently used to assist in reproducing the original x. P should be constructed in a way that even knowing its value would not affect uniform randomness of the generated value x. In terms of practical implementation we can not vouch for complete uniformity, yet we expect the string to be $\pm\epsilon$ close to uniform, where ϵ is exponentially a very small value.

Due to the differences between ω and ω' we need a way to firstly measure the extent of it and upon that fix the existing errors. The protocol proposal and its mathematical base for development logically imply the usage of error-correction codes since they are well-known and used for retrieval of original data in communication

systems. With the help of the generated, non-secret side string P and the input ω' we are to successfully extract the original and first input ω as well as the uniformly random generated string x. Dodis et al. [DRS04] claim that x extracted from ω may be used as a secure key in cryptographic application but with the difference that the key does not need to be stored (since it can be extracted again from the similar ω'. Therefore, the *fuzzy extractors* are information-theoretically secure and they may be used in crypto-systems without any further assumptions. However, note that this does not imply the cryptographic application itself to be computationally secure, which should be achieved by careful design and choice of parameters (length of secure keys, etc.)

In the next several sections we cover different terms that are used in the construction of our fuzzy commitment protocol. Mathematical theorems along with some constructions are explained with detailed taxonomy introduction of distance metrics, random number extractors, hash functions, entropy, code spaces along code words and the implications/application of error correction codes.

4.1 Mathematical definitions and terms

All of the calculations in our fuzzy commitment protocol are made under a defined metric space. By definition, a metric space[1] is a set M with a defined distance function $d : M \times M \leftarrow \mathbb{R}^+ = [0, \infty >$, where all distances between members of the set are fixed. The set of these distances is called a metric of the set [Bry85]. M itself is a finite set, bounded by the number of different values that a string consisted in a set can have (depends on its size, more particularly on the number of bits in binary form). The distance function d on set M can only take integer values and respects following characteristics:

1. distance of two same strings is zero, $d(a, b) = 0$, if and only if $a = b$,
2. distance between two strings is symmetrical, $d(a, b) = d(b, a)$, and
3. distance conforms to the law of triangle inequality, $d(a, c) \leq d(a, b) + d(b, c)$.

Along with the various distance functions we might use the term of *Hamming weight* of a string which gives the number of non-zero characters in it[Wei91].

4.1.1 Metrics used in fuzzy commitment protocol scheme

Our protocol is based on three different metrics that appear natural to use given the design of fuzzy commitments. Hence, we introduce the definitions of those metric as follows [DRS04]:

[1]For easier understanding, one very familiar metric space is the 3-dimensional Euclidean space where we often use the term *metric* as a generalization of Euclidean space metric well known as the Euclidean distance.

1. **Hamming metric** defines the distance function $d(a, b)$ as the number of positions in which the two strings a and b are different. The metric space is defined as $M = \Omega^n$, where Ω is some alphabet of defined size (different values), and n is the size of the strings for subsequent comparison. Additionally, we might use the term of weighted Hamming metric which differs from the original one by the distance calculation between specific symbols of the alphabet Ω. For example, if the alphabet is consisted of symbols $1, 2, 3, 4$ the distance between two subsequent symbols $(1, 2)$ may be one while the distance between other symbols is different $(d_w(1, 3) = d_w(2, 4) = 2)$,
2. **Set difference metric** represents the size of the symmetric difference of the two input sets ω and ω'. Let us assume that there exists a universe U of possible features. Our metric space M then consists of all subsets of the universe U. The symmetric difference of the two compared sets may be presented as $\omega \triangle \omega' \triangleq \{x \in \omega \cup \omega' | x \notin \omega \cap \omega'\}$. Therefore, the symmetric difference is equal to $|\omega \triangle \omega'|$. In our constructions we use a sub-method of the set difference metric as well. Due to the different sizes of elements in a specific universe we might want to explore the differences only between set elements of the same size. Hence, the metric space M is then going to be restricted to metric space M_k to contain k-element subsets, where k is the size of symbols in a string.
3. **Edit metric** is somewhat similar to the original Hamming distance metric. The metric space is $M = \Omega^*$, where Ω is the same alphabet mentioned above. Unlike Hamming distance, in edit metric the distance between ω and ω' is calculated by the minimum number of symbol insertions and deletions needed to transform the string ω to ω'. Another difference in the edit metric is that inserting or deleting a character from the string shifts the rest of it to the right or left, respectively. In our protocol, we use Levenshtein distance metric ([YB07]) which is further and more closely discussed in Chapter 3.

4.1.2 Codes and correcting mechanisms

The mechanism that we use in our fuzzy commitment protocol implies the need for correcting errors that are formed as a consequence of similarity (and not exact equality) between the two device inputs ω and ω'. One obvious solution is utilization of error-correction codes which, depending on their type and structure, may fix a defined number of differences between two string generated from the input. In order to do so we have to introduce the *code* and *syndrome* terms.

A code, marked as C, is a subset $\{\omega_0, \ \omega_1, \cdots, \ \omega_{m-1}\}$ of m elements of M. The process of mapping symbols/substrings from the sequence number i to the exact element ω_i is called encoding. We use this term in the thesis very often to implicate processing of the original fuzzy input (in our case user drawings) into meaningful strings of some alphabet Ω. Since we calculate the distance as an integer,

we define the minimum distance of the code C as the smallest $d > 0$, such that $\forall i, j \in 0 \cdots m, i \neq j$ the distance $d'(\omega, \omega') > d$. Hence the error correction mechanism may detect a maximum of $(d-1)$ mistakes in an element of metric space M.

On the other hand, the number of detected errors is not equal to the number of errors that can be fixed. That number may defined as the error-correcting distance (ECD) t [DRS04] of code C. Thus, ECD is the largest number $t > 0$, such that $\forall \omega \in M$ there exists a maximum of one codeword c (from the set of codes C) where the radius representing the distance is of maximum value t around a string ω; distance $d(\omega, c) \leq t$ for at most one $c \in C$. Hence, the formed error-correction code can fix up to t errors in a string $\omega \in M$.

The process of finding the right codeword $c \in C$ when given the string ω, and with the condition that $d(\omega, c) \leq t$, can be intuitively called *decoding*, as proposed by Dodis et al. [DRS04]. The codeword for each string is therefore to be unique which is important when *recovering* the original ω from ω' on the other side. Due to the restriction posed by limiting the distance to integer values, and by the laws of triangle inequality we are guaranteed the maximum number of fixed error can be $t \geq \lfloor \frac{d-1}{2} \rfloor$. Codes in general are often denoted as $(M, m, d) - codes$, but due to our specific application (intention of fixing differences) of the codes in the error-correcting mechanisms we denote our formed codes as $(M, m, t) - codes$ in the rest of the thesis.

Syndromes

To successfully recover the original string ω from the codeword we need to determine the error pattern, denoted as e. The error pattern may be perceived as the string of the same size as the original ω and is equal to $e = \omega' - \omega$, where the $'-'$ operation indicates modulo addition in a finite field. It is important to mention that in this thesis we base our calculations and models on finite fields and linear codes[2].

The error pattern may be calculated using syndromes. A "syndrome of a vector is its projection onto subspace that is orthogonal to the code and can thus be intuitively viewed as the vector modulo the code" [DRS04]. In linear codes we can create a parity check matrix H with the properties that its rows generate the orthogonal space $C^{\perp}$. Then for any vector $v \in \Omega^n$ we calculate the syndrome as $syn(v) \triangleq Hv$. Important thing to note is that $v \in C \Leftrightarrow syn(v) = 0$ and that the parity matrix size depends on the number of bits in the metric space and the dimension of the code. For example, in the Hamming distance metric over Ω^n we can calculate the dimension of the code as $k = \log_{|\Omega|} m$. Following the notation for error-correction codes in the literature we may then denote our codes as $(n, k, d = 2 * t + 1)_{\Omega} - codes$[3].

[2] Ω is a field, Ω^n is a vector space over Ω, and lastly the code C is a linear subspace of it

[3] The maximum possible value of m may be denoted as $A_{|\Omega|}(n, d)$ where the base is likely to be omitted if $|\Omega| = 2$. The code's metric space is then obviously set over $\{0, 1\}^n$

Hence, the parity check matrix is an $(n-k) \times n$ matrix and the syndrome of any vector is $n-k$ bits long.

We calculate the syndrome due to its properties of capturing all the necessary information for decoding and hence recovering the original information. Let us assume an example where the codeword c is sent to the other side and the string $c' = c + e$ is received. Then the syndrome of c' is exactly equal to the syndrome of the errors since it stands that $syn(c') = syn(c) + syn(e) = 0 + syn(e)$. In regards to metric properties, for any value vector v there exists a maximum of one vector e of weight less than $\frac{d}{2}$ such that it is equal to the syndrome of the error[4].Therefore, calculating the syndrome of the received vector is enough to determine the pattern of errors e and thus fix mistakes[5].

4.1.3 Entropy calculations

The goal of our device pairing protocol is secure bootstrapping of the communication line between two device. Hence, the security of device communication depends on the protocol design. Since we use random numbers in the protocol, first for creating a secret value and afterwards delivering it to the second device, we need to know up to what extent/certainty the potential adversary can predict generated random values, and thus acquire the secret key. Depending on the number of different values that a random number can have, one of the possible attacks is the guessing attack. If the random number is too small the adversary has a bigger probability to guess it correctly.

Predictability of a random variable X can be represented as $max_a\ Pr|X = x|$, while the value that we are interested in is its entropy:

$$H_\infty(X) = \log(max_a\ Pr|X = x|) \tag{4.1}$$

For security proofs we are always interested in worst case scenarios, where in this case it is the lowest possible entropy that can be achieved [CG88]. If we calculate the minimum entropy of some distribution we get the number of almost uniform random bits that can be extracted from it. We use the term "almost" uniform due to the limitations of measuring and applying algorithms that would give us pure uniformity. Nisan et al. [NZ96] defined the term *randomness extractor* function as:

– Let $Ex : \{0,1\}^n \leftarrow \{0,1\}^l$ be a probabilistic function on polynomial time that uses r bits as the seed for randomness. Then we can define Ex to be an

[4]This is valid due to the impossibility of the fact that given two distinctive vectors v_1 and v_2 their modulo addition/subtraction would give a codeword of weight less than d

[5]Note that the number of errors $|e| \leq t = 2 * d + 1$, or the original vector cannot be recovered

efficient $(n, m, l, \epsilon) - strong\ extractor$ if for all min-entropy m distributions W on $\{0,1\}^n$, it follows that:

$$\mathbf{SD}((Ex(W;X),(U_l,X)) \leq \epsilon \tag{4.2}$$

where U_l denotes the uniform distribution on $l-bit$ binary strings, X is uniform on $\{0,1\}^r$, and **SD** denotes the statistical distance between two probability distribution expressed as:

$$\mathbf{SD}(A,B) = \frac{1}{2}\sum_v |Pr(A=v) - Pr(Bs=v)| \tag{4.3}$$

According to the Radhakrishnan et al. [RTS00] a strong extractor may extract $l = m - 2\log(\frac{1}{\epsilon} + O(1))$ random bits at most. We do not present the proof since it is irrelevant for the main topic, yet we seek a plausible extractor construction. However, constructing a proper extractor may often be too complex. Thus, according to the Carter et al. [CW77] and Wegman et al. [WC81] who explored the properties of the universal hash functions and to Dodis et al. [DRS04] who resolves a proof of the usage of hash functions as strong extractor we get the optimal number of bits as:

$$l = m - 2\log(\frac{1}{\epsilon}) + 2 \tag{4.4}$$

4.2 Defining sketches and extractors for fuzzy input handling

To construct our fuzzy device pairing protocol we use a certain combination of techniques and sketches proposed in Dodis et al.[DRS04]. The idea of the protocol design is based on two simple constructions: i) Secure sketches, and ii) Fuzzy extractors. *Secure sketch* is a secure construction that allows precise reconstruction of the noisy input in such a way that having input ω the procedure outputs a sketch s. That sketch is used on the other side where if given the input $\omega' \backsim \omega$ it is possible to completely recover the original input ω. Security of the sketch relies in the masquerading of the input ω in a way that the sketch s does not reveal much about the original input. Hence, ω retains the majority of its entropy even if the sketch is known to the adversary. Unlike the secure sketch, a fuzzy extractor does not deal with the reproduction of the original input, yet it is oriented on creating a uniformly random secret. On one side the extractor takes the input ω and reproduces two vectors, P and R while on the other side the extractor takes the created vector P and the input $\omega' \backsim \omega$ which results in the construction of the same vector R which can be used as a key in a cryptographic application. Both parties can verify the success of the generation of R by checking that they have the same $f(R)$, where we need f to be a one-way collision resistant function. In further subsections we

explain the formal mathematical definitions of sketches and extractors as presented in [DRS04] with some modifications in notation that will suit our protocol design further in the thesis.

4.2.1 Formal definition of a secure sketch

As previously, let M be the metric space with a defined distance function d. Then we can define $(M, m, \widetilde{m}, t) - \textit{secure sketch}$ as a pair of randomized procedures to create and recover the sketch (S and S', respectively). The sketch possesses the following properties:

1. When given the noisy input $\omega \in M$ the first (sketching) randomized procedure returns a bit vector $s \in \{0,1\}^*$,
2. When given another noisy input $\omega' \in M$ and a bit vector $s \in \{0,1\}^*$ the *correctness* feature of the secure sketch guarantees that if $d(\omega, \omega') \leq t$, then the $S'(\omega', S(\omega)) = \omega$. However, if the $d(\omega, \omega') \geq t$ than the recovered vector $\omega_{S'}$ is unpredictable and incorrect,
3. The secure sketch enjoys the *security* feature that assures for any distribution W over the metric M, with the minimal entropy denoted as m that the value of W can be recovered by a malicious adversary, who has knowledge of s, with the maximum probability of 2^{-m}. Thus, $\widetilde{H_\infty}(W|S(W)) \geq \widetilde{m}$, and
4. Both of the randomized procedures are considered efficient if they run in polynomial time.

However, the security properties of the secure sketch also depend on the input ω, since it is not defined in the sketch how the ω is created. It may be that a potential adversary has some probabilistic information i about the input ω in a way that it can reveal a lot of information about the original input. In general, the input is hard to predict even given the additional information, but due to these circumstances the simple definition of the secure sketch cannot stand since there is no assurance that $H_\infty(W|i)$ is fixed to at least m bits. Therefore, a more robust definition whose security properties as strengthened is as follows: Let M be a metric space with a defined distance function d and $(M, m, \widetilde{m}, t)$ a secure sketch with additional features implying that for any random variables W over M and I over $\{0,1\}^*$ such that $\widetilde{H_\infty}(W|I) \geq m$. Then we have $\widetilde{H_\infty}(W|(S(W), I)) \geq \widetilde{m}$.

In both definitions the notation of $\widetilde{m}$ defines residual minimal entropy. The value may be calculated as $\widetilde{m} = m - \lambda$, where λ is the entropy loss. The value of entropy loss is very important in the security analysis of designed protocols where we aim to restrain it and provide some upper bound for which the protocol is secure. Thus, for a given construction of S and S' with the given value t we get the entropy loss λ such that for any value of m the (S, S') is a $(M, m, m - \lambda, t) - \textit{secure sketch}$.

4.2.2 Formal definition of a fuzzy extractor

As previously, let M be the metric space with a defined distance function d. Then we can define $(M, m, l, t, \epsilon) - fuzzy\ extractor$ as a pair of randomized procedures to generate and reproduce the value (*Gen* and *Rep*, respectively). The sketch possesses the following properties:

1. When given the noisy input $\omega \in M$ the first (generating) randomized procedure returns an extracted vector $R \in \{0,1\}^l$ and a helper vector $P \in \{0,1\}^*$,
2. When given another noisy input $\omega' \in M$ and a helper vector $P \in \{0,1\}^*$ in the reproducing construction of the extractor the *correctness* feature of the fuzzy extractor guarantees that if $d(\omega, \omega') \leq t$ and R, P are generated by $(R, P) \leftarrow Gen(\omega)$, then the $Rep(\omega', P) = R$. However, if the $d(\omega, \omega') \geq t$ than the recovered vector $R_{\omega'}$ is unpredictable and incorrect,
3. The secure sketch enjoys the *security* feature that assures for any distribution W over the metric M, with the minimal entropy denoted as m that the extracted vector R is nearly uniform even if the helper vector P can be observed by the potential adversary. Additionally, if $(R, P) \leftarrow Gen(W)$, then $SD((R, P), (U_l, P)) \leq \epsilon$, and
4. Both of the randomized procedures are considered efficient if they run in polynomial time.

Thus, a fuzzy extractor may be seen as a construction that is used to extract a random vector R from ω (and maybe another input depending on the protocol design and application) and afterwards the same vector may be reproduced from any vector ω' if it is close enough to the first input ω. During the generation process the extractor outputs a helper vector P that does not have to be secret since the fuzzy extractor design implies achieving truly random values of R even with the presence of vector P. However, the fuzzy extractor does not output truly uniform and random number, yet nearly uniform. This nearly uniform value can still (regarding properties of the extractor) used in a cryptographic application that requires completely uniform random bits (key establishment). Note that this limitation reduces security up to the distance ϵ from pure uniformity. If the distance ϵ is negligibly small, the losses are as well negligible and irrelevant.

4.3 Exemplar constructions for different metric spaces and distance functions

[Exemplar constructions]

In the next several subsections we define some constructions for different metrics as presented in relevant literature (note that the general literature overview is in the Chapter 2, while here we use some of the real examples). These constructions provide baseline elements for designing the fuzzy device pairing protocol further

on. Moreover, the constructions provide deeper understanding on secure sketches and fuzzy extractors since they are applicable solutions and not only theoretical definitions.

4.3.1 Hamming distance metric

Let the metric space be defined as $M = \Omega^n$, its size as $F = |\Omega|$ and $f = \log_2 \Omega$. The distance is calculated based on the Hamming distance metric. To create a secure sketch we may use code-offset construction [DRS04]. On one side, take the input ω and select a random codeword c [6]. The secure sketch of the input, $S(\omega)$, is set to be the shift needed from the input to the codeword c calculated as: $S(\omega) = \omega - c$, where if we have $\Omega = \{0, 1\}$ then the subtracting operation is equal to the modulo (addition) operation. The recovery procedure is then carried out by subtracting the shift s from the second input ω' to get $c' = \omega' - s$. If $d(\omega, \omega') \leq t$ then according to the performed operations, we also have $d(c, c') \leq t$. Hence, decoding procedure of c' to acquire c can be done successfully. Now when the original codeword c is calculated and we have received the shift s it is easy to recover the original input ω as $\omega = c + s$[7].

First construction using code offsets was introduced by Juels et al. [JW99] as a fuzzy commitment scheme where the secrets were not revealed in the same fashion as other commitment protocols. The definition of the construction may be seen as $(\Omega^n, m, m - (n - k)f, t)$ secure sketch. The entropy loss depends on the added k random bits[8] of Ω and sent n bits as the input. Therefore, given a $(n, k, 2t + 1)$ error correction code the secure sketch would be efficient in its encoding and decoding procedures. Additionally, since we focus on linear codes the output is fixed and is $(n - k)$ bits long. Accordingly, we can calculate the entropy loss as $nf - \log_2 A_\Omega(n, 2t + 1)$[9] if the input is nearly uniform and $m = nf$ since $K(M, t)$ is equal to $A_\Omega(n, 2t+1)$. For optimal codes the offset construction is as well optimal and then the entropy loss is $nf - \log_{|\Omega|} K \log_2 |\Omega| = nf - \log_2 K$. However, the creation of the efficiently decodable code remain a great challenge due to its dependency of $|\Omega|, n$ and d.

For the creation of fuzzy extractors, which will be one of the main design properties of our fuzzy commitment scheme, we construct the same code offset denoted as $v = \Omega - C(x)$. The first randomized procedure $Gen(\omega)$ then return $R = x$ and $P = v$. On the other side, the input from the second device and the helper string are used as

[6] Similar construction could be to choose a random number $x \in \Omega^k$ and then compute the corresponding codeword $C(x)$

[7] Note that we use almost the similar construction in the fuzzy device pairing protocol to extract the input from the first device in the second one

[8] The notation is in bits since $\Omega \in \{0, 1\}$. For other spaces we would use the term *symbols*.

[9] A_Ω denotes the maximum number of codewords K in a code with n bits and of distance d from an alphabet of size $F = |\Omega|$

$Rep(\omega', P)$ where the $C(x)$ is calculated as $\omega' - P$. If the distance $d(\omega, \omega') \leq t$ then we can successfully perform C^{-1} operation to get x. The extractor $(\Omega, nf, kf, t, 0)$ works well if the vector $P = v$ is truly random and independent of x when ω is random. This construction exploits the exact idea by Juels et al. [JW99] where they used the commitment process between devices to share and confirm knowledge of x, but without revealing it.

4.3.2 Edit distance metric

Let the metric space be defined as $M = \Omega^*$ for some alphabet Ω and its size as $F = |\Omega|$. The distance function between two vectors (or strings for non-binary alphabets) is defined as the smallest number of substitution, insertions and deletions needed to transform one vector to another. Note that due to the properties of the distance function the metric does not have transitive properties and the calculations are different then in the case of Hamming distance metric.

Due to the properties of our device pairing protocol related to synchronized drawing and comparison of the two pictures via different metrics we may expect high-distortion between the strings, as the metric results have shown in Chapter 3. Dodis et al. [DRS04] have conducted several measurements based on the properties of the metric space and strings themselves. They found out that the characteristics of metrics depend on the level of distortion between the compared strings. We will concentrate on the high-distortion features where according to the authors the entropy loss is roughly $2.38\sqrt[3]{tn \log n}$. Experimental results show as well that the number of corrected errors could be roughly $t < \frac{n}{15 * \log^2 n}$.

The construction for the edit distance metric is as follows [Bro97][OR07][GXTL10]. The idea is to construct shingles based on the original input and then compare them. A *c-shingle* is consecutive substring of the given input ω with the length of c. The result of a *shingling*[10] that processes a string ω of length n is a set of all $(n - c + 1)$ *c-shingles* of ω that are not particularly order and are without duplicates. Hence, the shingling results include all non-empty subsets of maximum size $(n - c + 1)$ over Ω^c. The output of the shingling process is denoted as $SH_c(\omega)$. Additionally, this construction exploits the usage of both set and edit distance in the following way. Let $\omega, \omega' \in \Omega^n$ be such that their distance $d(\omega, \omega') \leq t_1$. Let I be the number of actions of maximum t_1 insertions and deletion that transform one string to another. Each character deletion or insertion will add at most $(2c - 1)$ to the symmetric difference between two shingle sets, $SH_c(\omega)$ and $SH_c(\omega')$. Thus, the calculated distance is $d(SH_c(\omega), SH_c(\omega')) \leq (2c - 1)t$.

[10] For example, a 5-shingling process of a string "123456789" is {12345,23456,34567,45678,56789}

One example construction is as follows. Let $\omega \in \Omega^n$. We compute $SH_c(\omega)$ and afterwards store the resulting shingles in a lexicographic order $h_1, \cdots, h_k$, where $k \leq n - c + 1$. Second step is to naturally divide ω into $\lceil n/c \rceil$ c-shingles $s_1, \cdots s_{\lceil n/c \rceil}$ where there are no overlaps between the shingles except for the last two (since the string length is maybe not dividable by c) which overlap by maximum $c * \lceil n/c \rceil - n$ characters. Further on, for $1 \leq j \leq \lceil n/c \rceil$, set p_j to be the index $i \in \{0, \cdots, k\}$ such that $s_j = h_i$ (p_j tells the index of j th divided shingle of ω in the order set $SH_c(\omega)$. Then we define $g_c(w)$ to be the set containing p values as $g_c(\omega) = (p_1, \cdots, p_{\lceil n/c \rceil})$ with the maximum number of possible values equal to $(n - c + 1)^{\lceil n/c \rceil}$. Due to properties of this construction the original value of ω can then be recovered by knowing $SH_c(\omega)$ and $g_c(\omega)$. An exact fuzzy extractor may be created with an addition of a uniformly random number x which can, in addition to the $g_c(\omega)$, be exchanged in a helper string. The check by both parties to confirm the successful extraction is by comparing shingle sets that they acquired (first one uses the original set while the other recovers the original set with the help of its own set and the helper string). The values are to be hashed to avoid leaking of the information to a potential adversary.

An example construction for a secure sketch is somewhat similar. For the first randomized procedure $S(\omega)$ we can compute $v = SH_c(\omega)$ and calculate syndrome $s_1 = syn(x_v)$, where x is a generated random number. Furthermore we compute $s_2 = g_c(\omega)$ where each p_j is a string of $\lceil \log n \rceil$. From the value s_1, s_2 the output $s = (s_1, s_2)$ is created and transmitted to the other party. Upon receiving the value, the second randomized procedure $S'(\omega', (s_1, s_2))$ is initiated to recover the original v, sort it and then the original string may be easily recovered by concatenating shingle elements of v according to the values in s_2.

4.3.3 Set difference metric

The set difference metric is the least one connected to our design of the fuzzy device pairing protocol, but since we use some elements it is useful to devote some space to it and explain its properties. The inputs from both devices can now be seen as subsets of a universe U, where the size will be $n = |U|$. We then define the metric space by $SD_m(U)$. Set difference metric represents objects by the list of its features[11]. Thus, the distance between two sets ω, ω' is the size of the symmetric difference, $d(\omega, \omega') = |\omega \triangle \omega'|$. Each set ω is represented as its characteristic vector in binary form, $\{0, 1\}^n$, where 1 are at positions where $x \in U$ if $x \in \omega$, and 0 otherwise. Note that the settings of applicable solutions imply that the size n is much larger than the size of the observed set ω. Therefore, the representation of the set ω requires $|\omega| \log n$ bits.

[11] e.g. strings of various length in a long document, list of favorite objects, etc.

A good example of a construction for set difference metric is given by Dodis et al.[DRS04]. The authors sought to improve constructions for large universes made by Juels et al. [JS06]. Additionally, schemes from the information reconciliation literature by Minsky et al. [MTZ03] were analyzed as well. The resulting construction is as follows. Let us assume that size of n is one less than a power of two, $n = 2^m - 1$ for some chosen integer m. Then, n, m will identify U with the non-zero elements of the binary finite field of degree $m : U = GF(2^m)^*$. Furthermore, let $\omega \subseteq U$ of size s, where $n \gg s$. Let x_ω denote the characteristic vector of ω. According to the process of syndrome calculation we now have the sketch's first randomized procedure to be $S(\omega) = syn(x_\omega)$ which is $(n - k)$ bits long. Note that for smaller universe where $n \gg (n - k)$ there is a substantial improvement in the calculation time and efficiency. Hence, the use of $syn(x_\omega)$ as the sketch of ω implies that the code properties $(n - k)$ has to be very small. Maximum entropy of the input may be calculated as $\log(\binom{n}{s}) \approx s \log n$ and due to that the entropy loss on $n - k$ should certainly be under that value. The next step is to identify a suitable code for error correcting, where the family of binary BCH codes, introduced first by A. Hocquenghem in 1959 and then improved by Raj Bose and D. K. Ray-Chaudhuri [BRC60] in 1960, prove to be suitable. BCH codes are a family of $(n, k, 2t + 1)_2$ linear codes where $k = n - tm$ under the assumption that $n = 2^m - 1$. The code proves to be optimal for $t \ll n$ by the Hamming bound[12] where $k \leq n - \log(\binom{n}{t})$ [VLVL82]. Lastly, using the syndrome as a sketch with the constructed BCH code C, the entropy loss finally equals $n - k = t \log(n + 1)$.

This section concludes the chapter related to the mathematical background needed to comprehend and design the fuzzy device pairing protocol. The next chapter provides the exact protocol design and related assumptions.

[12]For any code of distance δ the circular radius of size $\lfloor(\delta - 1)/2\rfloor$ centered at different codewords must be disjoint. Each circular radius then contains $(\binom{n}{\lfloor(\delta-1)/2\rfloor})$ points and therefore $2^k * (\binom{n}{\lfloor(\delta-1)/2\rfloor}) \leq 2^n$. Specifically, the construction here implies that $\delta = 2t + 1$ and hence the boundary is at $k \leq n - \log(\binom{n}{t})$

Chapter 5
Protocol design

In this chapter we introduce the design of the synchronized drawing fuzzy device pairing protocol. The idea to use the synchronized drawing as a mechanism to establish authentication between two device was first introduced by Sethi et al. [SAA13] as a novel idea not previously used, at least from the available literature in the research community. Due to the uncertainty of the input entropy the authors have decided to use the commitment scheme rather than the fuzzy cryptography with error-correction codes. We continue research on the initial idea to create a fuzzy cryptography protocol with additional analysis on the metrics that could be used for drawing comparison. These results are presented in Chapter 3 where we researched a suitable metric that could be used in our design while taking into consideration the overall efficiency. Analysis resulted in a comparison between the metric and a journal paper published with the mentioned authors of the synchronized drawings commitment scheme protocol. The metric that we use in our design is the *LURD binary* metric. Even though the metric does not show the best result and is obviously worse than the original LURD metric, the LURD binary metric represents a viable solution in regards to its binary form. The problem with other metrics is the alphabet size, since the error-correction codes are mostly constructed to work with binary forms. Levenshtein's distance calculation, which is our primary comparison method of two fuzzy inputs, may work with all alphabet sizes but it is then more difficult to reconstruct original inputs if the alphabet is larger.

In the next several sections we explain the protocol design in detail by explaining all components, computations and exchanged message. We continue on with the construction of a suitable error-correction code that may fix the errors up to the metric boundaries for accepted and rejected drawings. Our goal is to eliminate any possibility of having false positives and maximally reduce the number of false negatives. Security analysis follows the design description where we comment on the viability of the protocol. Some uncertainties were identified, and thus, some solutions presented accordingly.

5.1 Protocol components and notation

The fuzzy device pairing protocol is divided into two stages on two separate sides, $device_1$ and $device_2$. In the first stage both of the devices acquire fuzzy inputs generated from drawings, they encode them and then $device_1$ "commits" his input value masqueraded with the random number to the $device_2$. After that, $device2$ reconstructs the original input and the generated random number from the first side to continue the protocol in the second stage. Second stage consist of operations that will enable $device_2$ to "commit" his input to $device_1$ using a different distance metric (i.e. set difference metric encompassed with the edit distance metric) for more enhanced security. In this way both of the device posses both inputs and a random number generated in the beginning. These values are enough to start performing a key generating mechanism that results in a shared key which can protect subsequent communication.

5.1.1 First protocol stage

To show all the functionalities of the protocol more closely we split the graphical representation of the whole protocol in 4 different parts (stage one and two along with the device-specific sides respectively). The complete picture may be seen in 5.5 the end of this Chapter. Additionally, all of the abbreviations, functions and variables of the protocol can be reviewed in Appendix A.

Figure 5.1 shows the first stage of the protocol on the side of $device_1$. The first step of the protocol is initiated when the user starts the pairing procedure on two devices. Application that we developed [1] capture the drawings in its raw form which is then encoded to suit the metric. After that (or at the same time) a suitable random number generator creates a random sequence x such that $x \in M$, $M = \{0,1\}^k$, where k is the length of the created sequence. Since the randomly generated number cannot be directly used for fuzzy cryptography (remember that we first have to define a code with certain distance properties in order to perform successful error correction) we translate the generated number into a specific codeword c, where $c \in C$, $C = \{0,1\}^n$, and n is the length of the codeword. The translation function is denoted with symbol g and can be represented as: $g : \{0,1\}^k \rightarrow \{0,1\}^n$, $g : M \rightarrow C$. The resulting codeword is then added to the encoded fuzzy input ω in modulo arithmetic to acquire the output of $device_1$'s first stage. The output P is exactly the helper vector that we discussed about in Chapter 4 under fuzzy extractors and it can be defined as: $P = c \bigoplus \omega$, $P \in \{0,1\}^n$.

[1] original code used from Sethi et al. [SAA13] with substantial modification, and a laptop touchpad surface apps developed from scratch

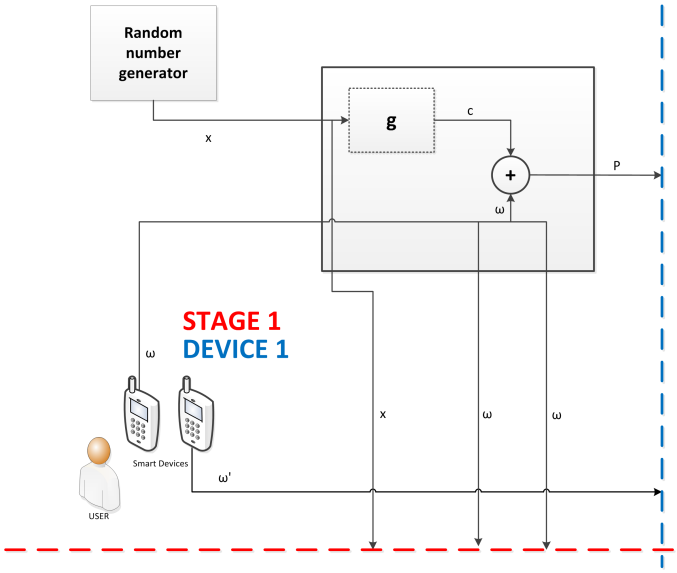


Figure 5.1: First protocol stage on the side of $device_1$

The next thing to observe is the first stage of the protocol on the side of $device_2$, represented by Figure 5.2. After the helper vector arrives on the device it is used to extract a codeword c' that should be close to the originally created codeword c. Namely, the received helper string P is added with modulo arithmetic to the input ω' in the same way as it was created: $P = c \bigoplus \omega' \equiv c' = P \bigoplus \omega'$, where $c' \in C'$, is the fuzzy extracted codeword and $C' \subseteq C$, $c' \in \{0,1\}^n$. The extracted fuzzy codeword c' acts as the input to the function denoted as f which is a decoding function (error-correction control) that translates c' to the closest matching codeword c. The recovered codeword will match the originally calculated codeword (computed on the side of $device_1$) if $d(\omega, \omega') \leq t$, where t is the maximum difference allowed in order to successfully extract c from c'. Thus, we define the functions as: $f : \{0,1\}^n \rightarrow C \bigcup \{\emptyset\}$. Basically, the function will either translate the codeword successfully under given limitations or the extracted codeword would not match and by performing the protocol operation further on, the result will be a failure in bootstrapping a secure connection.

If the original codeword is extracted correctly it is pretty straightforward to extract the input of $device_1$ by the $device_2$. Using the same operation of addition in modulo arithmetic the extracted codeword is combined with the received helper vector P which by the computation $P = c \bigoplus \omega \equiv \omega = P \bigoplus c$ returns the original input ω from the first device. This value is further on used in the second stage of the protocol so $device_2$ can prove to $device_1$ the successful decoding process. Along

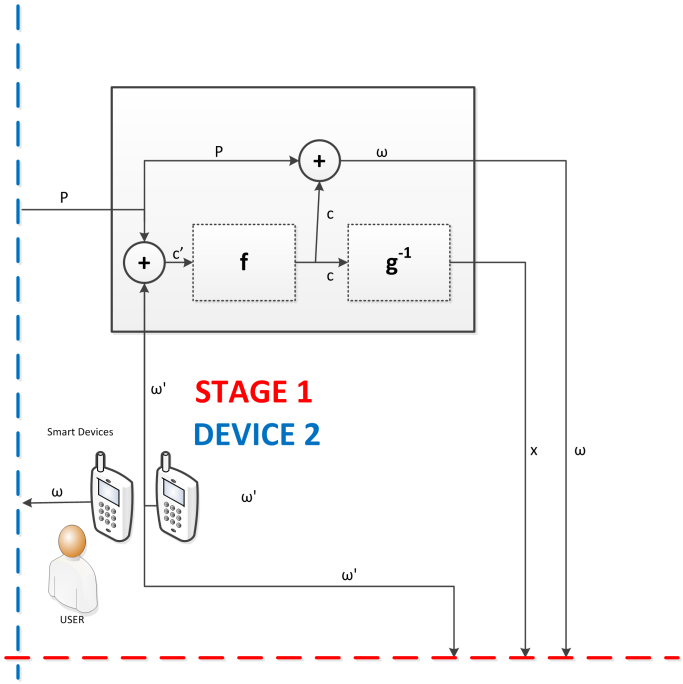


Figure 5.2: First protocol stage on the side of $device_2$

with the last operation, the reverse function g^{-1} of the codeword translation function g is performed in order to acquire the originally created random number x. This process should as well be straightforward if the codeword c is extracted properly and correctly. Random number x is used as well as original input ω further in the second stage of the protocol.

5.1.2 Second protocol stage

Unlike discussing the first protocol stage we will start from $device_2$ in this section. It is naturally to continue here since the first calculations are made on the second side. Figure 5.3 shows the second stage of the protocol on the side of $device_2$. As inputs from the first stage we may see the recovered original input ω, random number x from $device_1$ and the fuzzy input encoded on $device_2$, acquired in the beginning of the protocol bootstrapping operation. The first operation performed involves the characteristic set creation function $\delta(\omega)$ which takes the fuzzy input ω and outputs the c-shingle set. The operations for set/edit distance metric and the constructions of shingles were discussed in Chapter 4 under sections 4.3.2 and 4.3.3. Hence, we define the characteristic set creation function as: $\delta : \omega \leftarrow SH_c(\omega) \equiv \{\{0,1\}^s\}^{n-s+1}$, where s is the size of one s-shingle element that is a part of the characteristic set

with the total of $\lceil n/s \rceil$ elements. The $SH_c(\omega)$ is then used with the fuzzy input of the second device ω' as input to the characteristic shingle extraction function gc. Due to the similarities between ω and ω' we can allow this kind of usage in our protocol. Namely, the gc function, when using $SH_c(\omega)$, would only be used with ω as input. Our creation allows the usage of a similar input ω', when $d(\omega, \omega') \leq t$ since $SH_c(\omega) \cong SH_c(\omega')$. This procedure, as explained before, will allow $device_1$ to confirm successful operation on the side of $device_2$. Nevertheless, we define gc function as: $gc : (SH_c(\omega), \omega') \leftarrow \{0, \cdots, (n-s+1)\}^{\lceil n/s \rceil}$. The output of the function, denoted as $gc(\omega, \omega')$, is a sequence of ordinal numbers where the parts of the divided ω' can be found in the shingle set $SH_c(\omega)$.

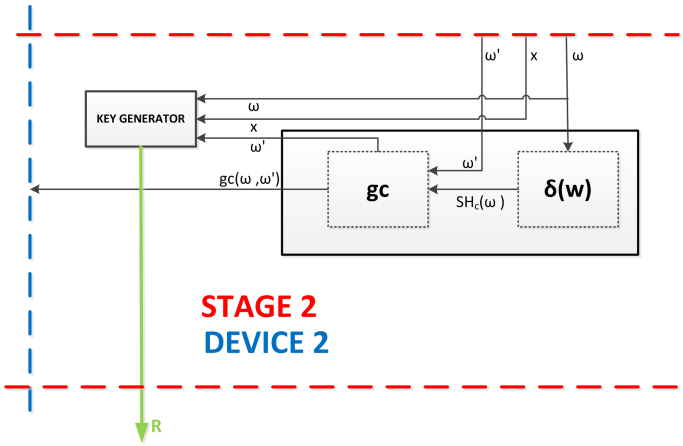


Figure 5.3: Second protocol stage on the side of $device_2$

After these protocol operations, $device_2$ is ready to initiate key generation process which is a variant of a hash or multiply nested hash functions for acquiring the key used to encrypt subsequent communication. The key generator takes ω, ω' and x as input and outputs the secret key. The protocol bootstrapping procedure is thus done in $device_2$.

Last step left is conducted on $device_1$ and is represented by Figure 5.4. As we may see, the first operation performed is the recovery of the fuzzy input of $device_2$. In the same fashion as in $device_2$ we perform the $\delta(\omega)$ function to acquire the shingle set $SH_c(\omega)$. The resulting set is then used as input to the inverse characteristic shingle extraction function gc^{-1} that basically takes the shingle set $SH_c(\omega)$ along with the sequence of parts ordinal numbers that, when put back together, can reproduce/recover the original fuzzy input. Obviously, we now have all the predispositions in $device_1$ to start the key generation process in the same way as in $device_2$. If all calculation during this bootstrapping have been encoded/decoded properly, two

devices will have the same secret key and can start meaningful conversation that depend on upper layer applications.

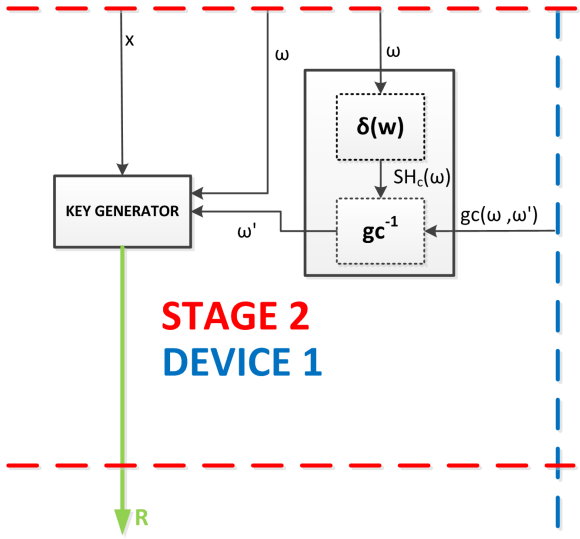


Figure 5.4: Second protocol stage on the side of $device_1$

Lastly, Figure 5.5 represents the protocol design in total where all of the four parts are connected. Brief descriptions of abbreviations, functions and variables are summarized in Appendix A.

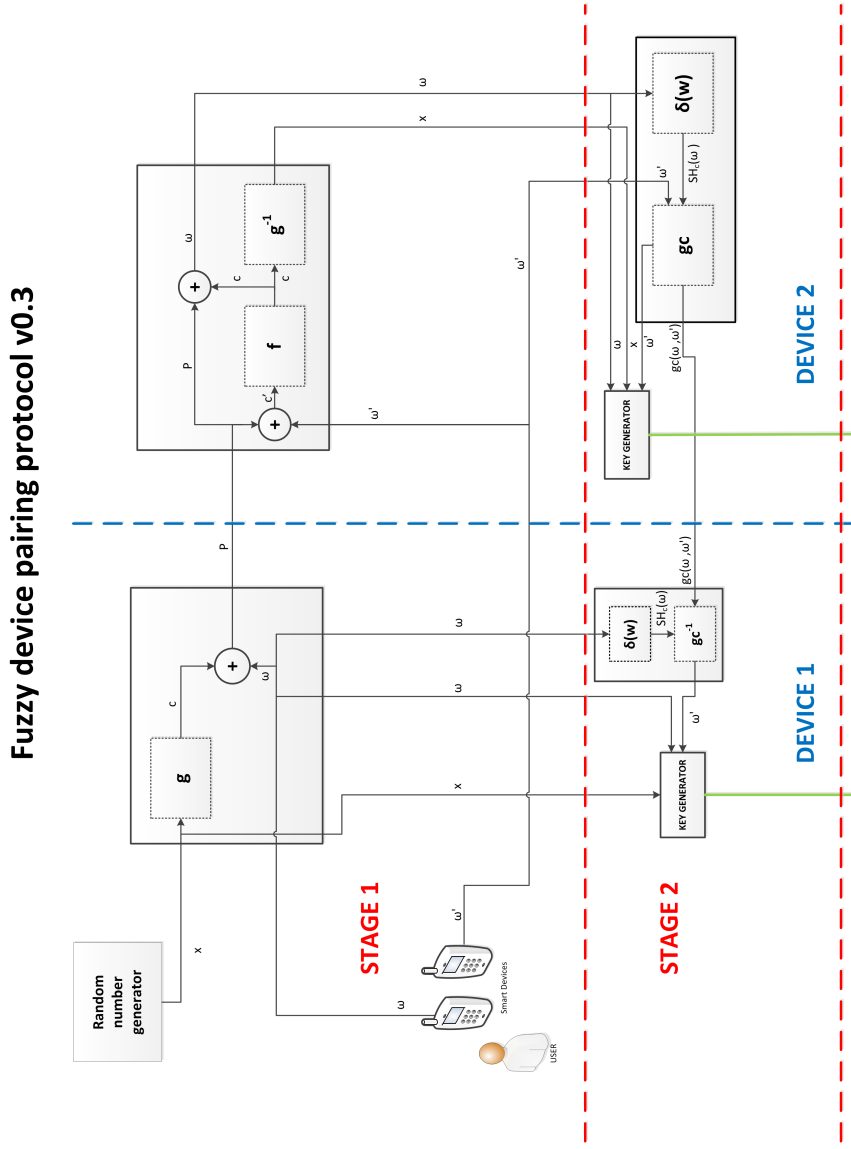


Figure 5.5: Complete design of the fuzzy device pairing protocol

5.1.3 Simplifying second protocol stage

In general, the complete protocol design could be simplified by completely removing stage two from the protocol. Practically, it is enough to deliver the original input ω to $device_2$ and use that along with the randomly generated number x to perform key generation, thus performing further authentication at a later stage after a shared secret is obtained. The disadvantage of this method is that it relies only on one random number generator and hence if one device is faulty there could not be a successful device pairing.

5.2 Construction of the code and error-correction mechanism

In this section our primary goal is to define the needed variables for our fuzzy device pairing protocol. Namely, we have to define the code length from which the codewords are picked up in relation to the generated random number, and as well define the shingle sizes that are used in the second protocol stage.

The first step is to observe the length of the fuzzy input generated by users' drawing on the touchpad surfaces. An average length of a LURD metric generated string from the input ranges from around 2500 symbols on average depending on the duration of the drawing and as well the number of movements. Furthermore, the users have to perform the drawing sequence for at least 4 seconds which gives us some lower bound. However, the upper bound is not strictly set and the users could presumably make drawings that would be even longer than e.g. 3000 symbols. In that case we will take only the first part into consideration and the rest is going to be discarded. We assume that these 2500 symbols will carry enough entropy (even though we cannot calculate it directly due to the properties of the fuzzy input) to perform bootstrapping of the secure protocol and derive a shared secret. In the end, we can imply reasonable security of the protocol if we are left with at least 128-bit entropy of the secret. Nowadays, that length of the secret is enough for many applications, and comparing it to the existing Bluetooth 4-pin security protocol (that only has the entropy of 13 bits) it is much more higher and reliable. However, in order to be able to perform an error-correction mechanism, codewords have to have redundancy, i.e. there has to be more symbols in the transmitted message than the actual information length to be able to reconstruct the original information from the input.

An extensive research throughout the coding theory and research community has given us the answer for choosing a suitable code that may fix the errors. I. S. Reed and G. Solomon published a paper in the Journal of Society for Industrial and Applied Mathematics in 1960 which describes a new class of error-correction codes

now called Reed-Solomon (RS) codes. Further on, Wicker et al. [WB99] performed a survey of the code properties and its applications. In general, Reed-Solomon codes are a group of linear cyclic error-correction codes that can detect and subsequently correct multiple random symbol errors. The mechanism is based on the addition of d check symbols to the original message which then enables detection up to any combination of d errors. However, the correction capacity is lower and equals a maximum of $\lceil d/2 \rceil$. The codes are very suitable for non-binary metrics since they can fix multiple bursts of bit errors (e.g. a sequence of $n + 1$ consecutive bit errors affect a maximum of two symbols that are n bits long). Nonetheless, our application uses binary symbols and is thus not affected by this property. The reason why we chose this code is its flexibility in determining the value d from within very wide limits. This allows us more freedom in the code construction which will in the end have better compliance with the design of the fuzzy device pairing protocol.

The principle of encoding is based on the idea that the original symbols (information) are viewed as coefficients of a certain polynomial $p(x)$ over a finite Galois field. The current implementations consider RS codes as a special case of cyclic BCH codes where encoded symbols are calculated from the coefficients of a defined polynomial constructed as the multiplication of a cyclic generator polynomial and $p(x)$. The original idea of RS codes encompasses creation of n code symbols from k symbols of the original information by oversampling the polynomial $p(x)$ at $n > k$ distinct point. These sample points are then transmitted and with the use of interpolation techniques the receiver is able to recover the original message [WB99]. The Reed-Solomon codes are popular today due to their great power and utility, and are found in many important applications such as computer electronics (CDs, DVDs, Discs in general, RAID), deep-space communications, data-transmission technologies and a wide array of broadcasting systems.

In this thesis we do not focus on actual encoding of the codewords which can be either related to the original idea of RS codes where a codeword is seen as a sequence of values or to the BCH codes point of view where a codeword is a sequence of coefficients. This problem relates directly to the code and error-correction mechanism implementation which is not the main goal here. However, the parameter setting is crucial and we address it. Firstly, it is important to explain the code variables and their construction. Reed-Solomon codes may be presented as a family of codes since for every parameter q, n and k there is an RS code with the alphabet size of q, block length $n < q$ and a message length of $k < n$. Its alphabet is represented as a finite field of order q, where the value of q then has to be a prime power. The most optimized RS codes follow the property which aims the block length to be a multiple constant of the message length denoted as $R = k/n$. Furthermore, the RS code possess a bound on the block length which has to be equal or one less than the size of the alphabet, hence, $n = (q|q - 1)$.

Formally, the RS code is a $(n, k, n-k+1)$ linear block code of length n over a field F with the dimension k and the minimum Hamming distance of $n-k+1$. RS code additionally complies to the Singleton bound, where it is basically optimal in a sense that its minimum distance has the maximum value for a linear (n, k) code[2]. This minimum distance actually determines the error-correcting capability of RS codes. Moreover, the minimum distance is the measure of redundancy in the block and if the location of erroneous symbols is not known beforehand then the Reed-Solomon code may correct up to $\lfloor (n-k)/2 \rfloor$ errors, which equals to the half of of redundant symbols added to the block of original information. In some cases, not applicable to out protocol design, the error locations might be known in advance. These errors are denoted as erasures and an RS code is able to correct a double amount of erasures. However, all the error positions will not always be known and the resulting error sequence might be a combination of errors and erasures. Thus, the RS code is then capable of correcting $2e + s \leq n - k$, where e denotes errors and s denotes erasures in the observed block. In practice, the RS codes are often constructed using a finite field F with 2^m elements where m represents the size of each symbol. Hence, on the sending side the blocks are encoded and each block has a size of $2^m - 1$[3].

In the end it is worth mentioning that the RS codes are transparent codes, similar to convolutional codes. This property allows the operation of the decoder even if the symbols were inverted along the transmission. However, RS codes are sensitive to shortening and thereby lose their transparency if the there are some missing bits in the decoder. We do not address this issue since we assume that the transmission channel between two devices that want to pair is clear and will not cause and additional modification of the originally created codeword on the side of the first device. These issues could be addressed upon real world implementation of the protocol since then we would have enough information about the channel properties and could measure potential modifications that are not directly related to the protocol design itself.

5.2.1 Setting the parameters for the RS code

As previously mentioned the only parameter that we know for now is the length of the fuzzy encoded input ω which is around 2500 symbols. Thus, we are able to set parameter k approximately, for now. Other parameters depend on the number of errors that our error-correction code is able to fix. To be able to determine the minimum distance for the code we have to look back in the Chapter 3 where different metric were analyzed. The LURD binary metric that we choose to use as our initial metric has a threshold on the $x - axis$ which basically tells us the percentage of differences discovered throughout drawing comparisons, and that number is exactly

[2]Such codes are also called Maximum distance separable (MDS) codes

[3]One of the very popular construction is the $(255, 223)_{RS}$ code where $m = 8$, $n = 2^8 - 1 = 255$ and the value of k, $k < n$ is 223 which basically allows the code to have 32 parity symbols and, thus, can correct up to 16 symbols per block.

our minimum distance which we have to be able to fix. If we look at the Figure 5.6 then it is easy to observe that the threshold for accepted drawings is 0.26 (calculated as the actual distance divided by the string length). This means that the value of t as the number of errors possible to fix is $0.26 * k$, and thus the minimum hamming distance $d = n - k + 1 = 2 * t$.

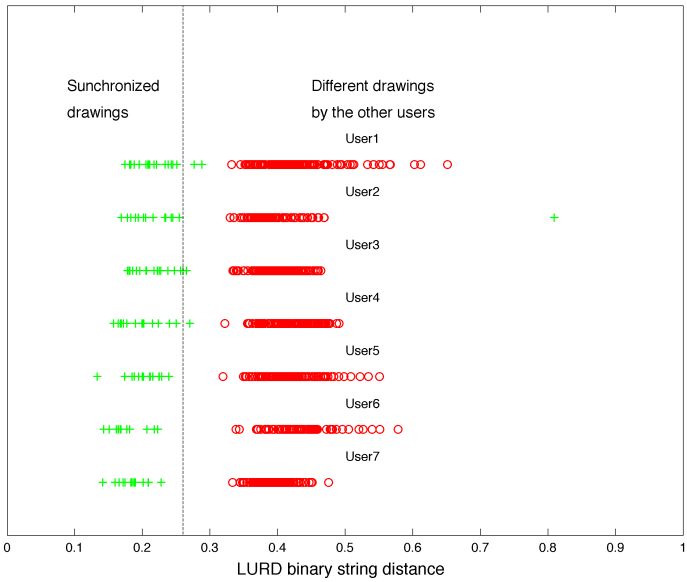


Figure 5.6: Comparison of metric distances for threshold evaluation

The goal is to find a set of parameters that will suit the equations posed by the RS code. If we set the value of m to be 12, then the value of the RS code alphabet is $q = 2^{12} = 4096$. Thus, the total length of the code is then $n = q-1 = 4096-1 = 4095$. Now we can acquire the actual value of allowed block length k from this equation:

$$n - k = 2 * 0.26 * k \leftarrow 4095 = (1 + 0.52) * k \leftarrow \lfloor k \rfloor = \frac{4095}{1,52} = 2694 \qquad (5.1)$$

Therefore, in our fuzzy device pairing protocol we will use a $(4096, 2694, 1403)_{RS}$ code. A useful thing to calculate furthermore is the entropy loss of this construction since the security of the protocol depends on it. If we refer to the Chapter 4, Section 4.1.3 the entropy loss can be calculated as $2.38\sqrt[3]{tn \log n}$ which equals approximately $\widetilde{774}$. This leaves is with more than enough entropy from the original fuzzy input. However, due to the inability to determine the actual entropy of the fuzzy input

(drawing characteristics are unpredictable) we are unable to determine the actual entropy that is left. A security threat for our protocol might be an adversary with a large statistical knowledge of drawn and encoded picture. Even though the number of recorded and saved pictures from the adversary would have to be exponentially large it is still considered a threat. Nonetheless, with the introduction of the random number which is incorporated in the creation of the codeword to masquerade the input we somehow overcome the issue. Let us assume that the random number generator on the side of $device_1$ is nearly uniform and since it is added in modulo arithmetic to the input it is practically infeasible to acquire statistical knowledge of the transmitted message which would make the protocol information-theoretically (note the Evaluation section that follows bellow on this topic) secure. In this protocol design we will, however, have some false negatives since some drawings cross the boundary of 0.26 distance.

5.2.2 Evaluating protocol and security properties

The protocol design explained in the above section unfortunately cannot be claimed completely secure without real-world implementation and experiments. Namely, there are several settings that might be ambiguous when considering the security of the protocol. Firstly, the construction based on Reed-Solomon codes uses a somewhat different metric for comparison and correction of the strings and therefore can achieve worse (or even better) results. Hence, the parameters of our RS code might be completely wrong and potentially would not fit with the sizes of the original framework. RS codes are mainly built on the basis of Hamming distance metric while in our case we focus the metric analysis on edit distance metric. Secondly, we do not posses any actual data on the error rates that the fuzzy inputs would create and to do so we would need a much larger user database with many more user test cases and experiments. An additional problem is the calculation of the entropy that is left after subtracting entropy loss based on the protocol construction. Since it is very difficult to calculate the entropy of the drawings then it is, naturally, difficult to subtract some value from it and further on claim the correctness of the result to be either secure or insecure.

We performed an approximate entropy calculation using cryptool[4] and based on the string length of 2694 symbols, the entropy per symbol and the compression ratio we end up with 843,2 bits of entropy. If we calculate the entropy loss by using the previously mentioned $2.38\sqrt[3]{tn\log n}$ we roughly get around 774.63 bits. By subtracting these value we can assume that the leftover entropy is 68.57 bit. Even though that might give enough security for a protocol of this type we cannot be certain of the calculation and protocol correctness.

[4] www.cryptool.org

However, the protocol design (if we disregard the comparison metrics and its application with synchronized drawing) is theoretically correct and supported by mathematical theorems from the widely available literature. Thus if we were to find a shared noisy environment and an input type that can be based and compared on the Hamming distance metric, we would be able to use the protocol immediately. One thing that would remain to determine is the entropy of the proposed inputs, and after subtracting the entropy loss, the leftover entropy which indicates the level of security.

Even though the design of the protocol and the code construction might leak some information to the adversary it is not known how the attacker could potentially use that information and revert the operations to recover the strings. Hence, the notation of "practical security" might be a potential term that is applicable to this case. It basically means that even though in theory a protocol or some secure communication is not proven to be theoretically secure, under given circumstances and parameters, it may be that they are still secure in practice due to difficult mechanisms of breaking them.

5.2.3 Using constructions for edit distance

Due to the discussed flaws we present a viable solution constructed by Dodis et al.[DORS08] in a paper *Fuzzy extractors: How to generate strong keys from biometrics and other noisy data* from year 2008 which is an extension of the same name paper by the same authors published in 2004. Their construction is directly related to acquiring a key based on input from a noisy environment. Moreover, the authors provide theorems and proofs for the construction and thus it may be used as a working example without further experimentation. The scheme may be applied to our protocol as a replacement for the Stage 1. Regarding the fact that the explanation is straightforward and that it is based on the mathematical background that we cover in the previous Chapter 4 we do not provide an explicit protocol diagram like for our fuzzy pairing device protocol. The construction is as follows.

Let us assume the existence of some space F^* for a defined alphabet F, where the distance between two strings is defined as the number of character insertions and deletions needed to transform one string to another. The metric space is of size n. Thus, we can construct a secure sketch $S(\omega)$ by computing $n - c + 1$ shingles $v = SH_c(\omega) = (v_1, v2, \cdots, v_{n-c+1})$, and $s_1 = syn(x_\omega)$, where the $x_v = \sum_{x \in \omega} x^i$ based on computations in $GF(2^m)$. Further on, we compute s_2 as $s_2 = g_c(\omega)$ by writing each p_j as a string of $\lceil \log n \rceil$. Hence, the output *helper* string that is transferred from $device_1$ to $device_2$ is denoted as $s = (s_1, s_2)$. In the $device_2$ the recovery procedure $S'(\omega', (s_1, s_2))$ is used to recover v by firstly calculating $SH_c(\omega)$ to acquire v', and then calculating $syn(v) = (v'_1 - x_{v(1)}, \cdots, v'_{n-x+1} - x_{v(n-c+1)})$.

Further procedure requires sorting v in the alphabetical order and then recovering the original ω by stringing along the elements of v according to the indices specified in the vector s_2. Both devices are now sharing the original input ω and can subsequently use it to generate a key using, for example, some type of hash function. The formal proof of the construction can be found in Dodis et al.[DORS08] under constructions 6-9. Note that in this construction the original LURD metric or even ANGLE metric can be used since the alphabet does not have to be binary (there are no error correction codes involved).

5.2.4 Determining the shingle set size

The shingle set size parameter refers to both constructions, the Stage 2 of the initial and main protocol design, as well as the additional subversion that covers the protocol under edit distance metric.

Unlike setting the parameters for the Reed-Solomon code, determining the size of the shingle sets is an easier task. We are looking for a shingle set that is large enough to cover all the possible combination that may occur in the fuzzy input of some size and additionally that it is not too large, since there might be differences in the shingle sets calculated from two different inputs ω and ω'.

Experimentally, we found a suitable shingle size to be 10 bits which equals to 5 original LURD symbols. If we look at the number of possible combinations it is equal to $4^5 = 2^{10} = 1024$. By taking more symbols the second protocol stage produces some inconsistencies and thus a shared secret cannot be successfully derived. On the other hand if we take less symbols then we open a path for the adversary since he can easily calculate all the possible shingle sets (256 different symbols) and with a good statistical knowledge about the inputs he can determine the ones that repeat very often. Thus, he might be able to reconstruct the fuzzy input ω' of $device_2$. The size of 1024 makes the guessing attack more difficult and the probability of the attacker to determine the input is very little. Tests with this shingle size have produced satisfactory results in more than 90% of the protocol bootstrapping instances in regards to successful delivery of the ω' to $device_1$.

With this section we finalize the protocol design. Ideas for further improvement and possibilities for future work might be found in the Conclusion chapter which follows further on.

Chapter 6
Conclusion

Throughout the thesis we have addressed several important questions regarding device pairing protocols. In the beginning we performed a literature overview of the current state of research regarding different ways and available constructions to design pairing protocols. Our focus moves to a specific type of device pairing protocols that lean on the term of *fuzzy cryptography* and hence we continued on investigating different approaches to design a fuzzy device pairing protocol based on synchronous drawing between two devices that have a touch surface. First part of the research involves extensive encoding and metric analysis for comparing two different, yet similar drawings. We conclude that the best quality can be achieved by using ANGLE and original LURD metrics when the edit distance metric is used. However, for our device pairing protocol we decided to use the binary version of the LURD metric since its form complies to the construction (even though it has a slightly worse performance).

The design of the protocol first started by discussing mathematical background and providing various constructions that may be used, some available from the literature and some designed as a combination of these. Even though the protocol constructed in this thesis was theoretically correct we have stumbled upon a limiting fact on using error correction code for its success. Namely, all of the constructions used in the fuzzy device pairing protocols were based on Hamming distance metric while the comparison metrics that we analyzed were based on Levenshtein's edit distance metric. Due to these difference we were unable to prove for certain if the protocol is secure enough and what are the actual performance parameters of the protocol. However, the protocol design and its mathematical construction gave us the ability to assume practical security since the reverse engineering process that would destroy its security was not straightforward to comprehend and achieve. Nevertheless, to provide a viable solution for the protocol based on synchronous drawing we use the constructions from the literature that was proved working and was based on the edit distance metric which complies to our metric analysis.

Further work on this topic encompasses the actual implementation of the first version of the device pairing protocol and analyzing its correctness and security. By doing this, we could with certainty comment on the protocol itself and possibly introduce modifications, if needed, in order for the protocol to achieve reasonable security in its domain.

References

[BBCM95] Charles H Bennett, Gilles Brassard, Claude Crépeau, and Ueli M Maurer. Generalized privacy amplification. *Information Theory, IEEE Transactions on*, 41(6):1915–1923, 1995.

[BBCS92] Charles H Bennett, Gilles Brassard, Claude Crépeau, and Marie-Hélene Skubiszewska. Practical quantum oblivious transfer. In *Advances in Cryptology—CRYPTO'91*, pages 351–366. Springer, 1992.

[BBR88] Charles H Bennett, Gilles Brassard, and Jean-Marc Robert. Privacy amplification by public discussion. *SIAM journal on Computing*, 17(2):210–229, 1988.

[BCN04] Claude Barral, Jean-Sébastien Coron, and David Naccache. Externalized fingerprint matching. In *Biometric Authentication*, pages 309–315. Springer, 2004.

[BD62] Richard Ernest Bellman and Stuart E Dreyfus. Applied dynamic programming. *Princeton, Nj. : Princeton Univ.*, 1962.

[BH03] Martin Berger and Kohei Honda. The two-phase commitment protocol in an extended pi-calculus. *Electronic Notes in Theoretical Computer Science*, 39(1):21 – 46, 2003. EXPRESS'00, 7th International Workshop on Expressiveness in Concurrency (Satellite Workshop from {CONCUR} 2000).

[Bla08] P. E. Black. Levenshtein distance. *Dictionary of Algorithms and Data Structures [online], U.S. National Institute of Standards and Technology*, 2008.

[BM92] Steven M Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 72–84. IEEE, 1992.

[BRC60] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and control*, 3(1):68–79, 1960.

[Bro97] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[Bry85] Victor Bryant. *Metric spaces: iteration and application*. Cambridge University Press, 1985.

[BSHL07] Daniel Bichler, Guido Stromberg, Mario Huemer, and Manuel Löw. Key generation based on acceleration data of shaking processes. In John Krumm, GregoryD. Abowd, Aruna Seneviratne, and Thomas Strang, editors, *UbiComp 2007: Ubiquitous Computing*, volume 4717 of *Lecture Notes in Computer Science*, pages 304–317. Springer Berlin Heidelberg, 2007.

[BSSW02] Dirk Balfanz, Diana Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Network and Distributed System Security Symposium (NDSS)*, 2002.

[CCH06] M. Cagalj, S. Capkun, and J-P Hubaux. Key agreement in peer-to-peer wireless networks. *Proceedings of the IEEE*, 94(2):467–478, Feb 2006.

[CDVdG88] David Chaum, Ivan B. Damgard, and Jeroen Van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology—CRYPTO'87*, pages 87–119. Springer, 1988.

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

[CS06] Amit K. Chopra and Munindar P. Singh. Contextualizing commitment protocol. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1345–1352, New York, NY, USA, 2006. ACM.

[CW77] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.

[DD09] Elena Deza and Michel Marie Deza. Encyclopedia of distances. In *Encyclopedia of Distances*, page 94. Springer, 2009.

[DFMP99] George I. Davida, Yair Frankel, Brian J. Matt, and René Peralta. On the relation of error correction and cryptography to an off line biometric based identification scheme. 1999.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and JanL. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*,

volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer Berlin Heidelberg, 2004.

[ED03] Carl Ellison and Steve Dohrmann. Public-key support for group collaboration. *ACM Trans. Inf. Syst. Secur.*, 6(4):547–565, November 2003.

[EHMS00] Carl Ellison, Chris Hall, Randy Milbert, and Bruce Schneier. Protecting secret keys with personal entropy. *Future Generation Computer Systems*, 16(4):311–318, 2000.

[FJ01] Niklas Frykholm and Ari Juels. Error-tolerant password recovery. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 1–9. ACM, 2001.

[Fre01] Gerhard Frey. *Applications of arithmetical geometry to cryptographic constructions*. Springer, 2001.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1987.

[GM12] Bogdan Groza and Rene Mayrhofer. Saphe: Simple accelerometer based wireless pairing with heuristic trees. In *Proceedings of the 10th International Conference on Advances in Mobile Computing*, MoMM '12, pages 161–168, New York, NY, USA, 2012. ACM.

[GMN04] Christian Gehrmann, Chris J Mitchell, and Kaisa Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7(1):29–37, 2004.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS*, volume 86, pages 174–187, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[GN01] Christian Gehrmann and Kaisa Nyberg. Enhancements to bluetooth baseband security. In *Proceedings of Nordsec*, volume 2001, pages 191–230, 2001.

[Gol96] I. Goldberg. Visual key fingerprint code. 1996.

[GR93] Jim Gray and Andreas Reuter. *Transaction processing: Concepts and techniques*. Kaufmann, 1993.

[GSS+06] M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun. Loud and clear: Human-verifiable authentication based on audio. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 10–10, 2006.

[GXTL10] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *Bell Systems Technical Journal*, 29(2):147–160, 1950.

[Har60] An implementation of syndrome encod- ing and decoding for binary bch codes, secure sketches and fuzzy extractors. 1960.

[HB01] Nicholas J Hopper and Manuel Blum. Secure human identification protocols. In *Advances in cryptology—ASIACRYPT 2001*, pages 52–66. Springer, 2001.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[JS06] Ari Juels and Madhu Sudan. A fuzzy vault scheme. *Designs, Codes and Cryptography*, 38(2):237–257, 2006.

[JW99] Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 28–36. ACM, 1999.

[JW01] Markus Jakobsson and Susanne Wetzel. Security weaknesses in bluetooth. In David Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 176–191. Springer Berlin Heidelberg, 2001.

[KB02] Serkan Kaygin and M.Mete Bulut. Shape recognition using attributed string matching with polygon vertices as the primitives. *Pattern Recognition Letters*, 23(1-3):287–294, January 2002.

[KFR09] Ronald Kainda, Ivan Flechais, and A. W. Roscoe. Usability and security of out-of-band channels in secure device pairing protocols. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, pages 11:1–11:12. ACM, 2009.

[KPDS02] V. Kumar, N. Prabhu, M.H. Dunham, and A.Y. Seydim. Tcot-a timeout-based mobile transaction commitment protocol. *Computers, IEEE Transactions on*, 51(10):1212–1218, Oct 2002.

[KSHW98] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. In *Information Security*, pages 121–134. Springer, 1998.

[KSTU09] A. Kumar, N. Saxena, G. Tsudik, and E. Uzun. Caveat eptor: A comparative study of secure device pairing methods. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1–10, March 2009.

[KSW07] Darko Kirovski, Michael Sinclair, and David Wilson. The martini synch: Joint fuzzy hashing via error correction. In Frank Stajano, Catherine Meadows, Srdjan Capkun, and Tyler Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks*, volume 4572 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin Heidelberg, 2007.

[Lan04] Michael Langberg. Private codes or succinct random codes that are (almost) perfect. In *FOCS*, volume 4, pages 325–334, 2004.

[Lev65] V. I. Levenshtein. Binary codes with correction of deletions, insertions and substitution of svmbols. *Dokl. Akad. Nank. SSSR.*, 163:845–848, 1965.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, inser- tions, and reversals. *Sov. Phys.—Dokl.*, 10(8):707–710, 1966.

[Lip94] Richard J Lipton. A new approach to information theory. In *STACS 94*, pages 699–708. Springer, 1994.

[LN06a] Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In David Pointcheval, Yi Mu, and Kefei Chen, editors, *Cryptology and Network Security*, volume 4301 of *Lecture Notes in Computer Science*, pages 90–107. Springer Berlin Heidelberg, 2006.

[LN06b] Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In *Cryptology and Network Security*, pages 90–107. Springer, 2006.

[Mae91] Maurice Maes. Polygonal shape recognition using string-matching techniques. *Pattern Recognition*, 24(5):433–440, January 1991.

[May07] Rene Mayrhofer. The candidate key protocol for generating secret shared keys from similar sensor data streams. In Frank Stajano, Catherine Meadows, Srdjan Capkun, and Tyler Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks*, volume 4572 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2007.

[MG07] Rene Mayrhofer and Hans Gellersen. Shake well before use: Authentication based on accelerometer data. In Anthony LaMarca, Marc Langheinrich, and Khain Truong, editors, *Pervasive Computing*, volume 4480 of *Lecture Notes in Computer Science*, pages 144–161. Springer Berlin Heidelberg, 2007.

[MMV+11] Suhas Mathur, Robert Miller, Alexander Varshavsky, Wade Trappe, and Narayan Mandayam. Proximate: Proximity-based secure pairing using ambient wireless signals. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 211–224, New York, NY, USA, 2011. ACM.

[MPR05] J.M. McCune, A. Perrig, and M.K. Reiter. Seeing-is-believing: using camera phones for human-verifiable authentication. In *Security and Privacy, 2005 IEEE Symposium on*, pages 110–124, May 2005.

[MPSW05] Silvio Micali, Chris Peikert, Madhu Sudan, and David A Wilson. Optimal error correction against computationally bounded noise. In *Theory of Cryptography*, pages 1–16. Springer, 2005.

[MRLW01] Fabian Monrose, Michael K Reiter, Qi Li, and Susanne Wetzel. Cryptographic key generation from voice. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 202–213. IEEE, 2001.

[MRW02] Fabian Monrose, Michael K Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. *International Journal of Information Security*, 1(2):69–83, 2002.

[MTZ03] Yaron Minsky, Ari Trachtenberg, and Richard Zippel. Set reconciliation with nearly optimal communication complexity. *Information Theory, IEEE Transactions on*, 49(9):2213–2218, 2003.

[MW07] R. Mayrhofer and M. Welch. A human-verifiable authentication protocol using visible laser light. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 1143–1148, April 2007.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[NR06] L. H. Nguyen and A. W. Roscoe. Efficient group authentication protocol based on human interaction. *In Proceedings of the Workshop on Foundation of Computer Security and Automated Reasoning Protocol Security Analysis (FCS-ARSPA)*, pages 9–33, 2006.

[NSHJ12] N. Nguyen, S. Sigg, A. Huynh, and Y. Ji. Pattern-based alignment of audio data for ad hoc secure device pairing. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pages 88–91, June 2012.

[NSHY12] Nguyen Ngu, S. Sigg, A. Huynh, and Ji Yusheng. Using ambient audio in secure mobile phone communication. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 431–434, March 2012.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 33–43, New York, NY, USA, 1989. ACM.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[OR07] Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *Journal of the ACM (JACM)*, 54(5):23, 2007.

[PS99] 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.

[PV06] Sylvain Pasini and Serge Vaudenay. Sas-based authenticated key agreement. In *Public Key Cryptography-PKC 2006*, pages 395–409. Springer, 2006.

[RS84] Ronald L Rivest and Adi Shamir. How to expose an eavesdropper. *Communications of the ACM*, 27(4):393–394, 1984.

[RSA78] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RTS00] Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, 2000.

[SA02] F. Stajano and R. Anderson. The resurrecting duckling: security issues for ubiquitous computing. *Computer*, 35(4):22–26, Apr 2002.

[SAA13] Mohit Sethi, Markku Anrikainen, and Tuomas Aura. Commitment-based device pairing with synchronized drawing (too appear in Percom conference. 2013.

[San72] David Sankoff. Matching sequences under deletion/insertion constraints. *Proceedings of the National Academy of Sciences*, 69(1):4–6, 1972.

[SEKA06] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6 pp.–313, May 2006.

[SGB67] Claude E Shannon, Robert G Gallager, and Elwyn R Berlekamp. Lower bounds to error probability for coding on discrete memoryless channels. i. *Information and Control*, 10(1):65–103, 1967.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sha02] Ronen Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[SJ12] S. Sigg and Y. Ji. Adhocpairing : Spontaneous audio based secure device pairing for android mobile devices. 2012.

[SS13] D. Schurmann and S. Sigg. Secure communication based on ambient audio. *Mobile Computing, IEEE Transactions on*, 12(2):358–370, Feb 2013.

[STU08] 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, *Information Security*, volume 5222 of *Lecture Notes in Computer Science*, pages 385–400. Springer Berlin Heidelberg, 2008.

[SVA07] Jani Suomalainen, Jukka Valkonen, and N Asokan. Security associations in personal networks: A comparative analysis. In *Security and Privacy in Ad-hoc and Sensor Networks*, pages 43–57. Springer, 2007.

[TK76] E. Tanaka and T. Kasai. Synchronization and substitution error-correcting codes for the levenshtein metric. *Information Theory, IEEE Transactions on*, IT-22:156–162, 1976.

[UKA07] Ersin Uzun, Kristiina Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 307–324. Springer Berlin Heidelberg, 2007.

[Vau05] Serge Vaudenay. Secure communications over insecure channels based on short authenticated strings. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 309–326. Springer Berlin Heidelberg, 2005.

[VLVL82] Jacobus Hendricus Van Lint and Jacobus Hendricus Van Lint. *Introduction to coding theory*, volume 86. Springer, 1982.

[VSLDL07] Alex Varshavsky, Adin Scannell, Anthony LaMarca, and Eyal De Lara. Amigo: Proximity-based authentication of mobile devices. In *UbiComp 2007: Ubiquitous Computing*, pages 253–270. Springer, 2007.

[WB99] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[WC81] Mark N Wegman and J Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.

[Wei91] Victor K Wei. Generalized hamming weights for linear codes. *Information Theory, IEEE Transactions on*, 37(5):1412–1418, 1991.

[WF74] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[YB07] L. Yujian and L. Bo. A normalized levenshtein distance metric. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1091–1095, 2007.

[Zad97] Lotfi A Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy sets and systems*, 90(2):111–127, 1997.

Appendix

Abbreviations, functions and variables of the fuzzy device pairing protocol

ω - represent the sensory input from device1

ω' - represents the sensory input from device2

x- represents a sequence from space $M = \{0,1\}^k$, $x \in M$,

where k represents the length of variable x

g - translation function to a codeword, $g : \{0,1\}^k \to \{0,1\}^n, g : M \to C$,

where $C = \{0,1\}^n$, $c \in C$, and n is the length of the codeword

g^{-1} - revert function for acquiring the generated random number from the codeword

c - represents the codeword, $c \in C$

P - represent the fuzzy (helper) vector that is sent from device1 to device2, $P \in \{0,1\}^n$, and $P = c \oplus \omega$

c' - fuzzy extracted codeword,

where $C' \subseteq C$, $c' \in \{0,1\}^n$, $c' \in C'$, and $P = c \oplus \omega'$

f - decoding function (error correction control) that translates c' to c if the $d(\omega, \omega') < t$, $f : \{0,1\}^n \to C \cup \{\emptyset\}$

where t is the maximum difference allowed in order to successfully extract c from c'

$\delta(\omega)$ - characteristic set creation function outputs $SH_c(\omega)$, $\delta : \omega \to \{\{0,1\}^s\}^{n-s+1}$,

where s is the size of one s-shingle element that is a part of the characteristic set with the total of $\lceil n/s \rceil$ elements

gc - characteristic shingle extraction function,

where $gc : (SH_c(\omega), \omega') \to \{0 \cdots (n - s + 1)\}^{\lceil n/s \rceil}$

output - ω' (presented as well as input), $gc(\omega, \omega')$ as sequence of ordinal number where the parts of the divided ω' can be found in the shingle set $SH_c(\omega)$

$KEYGENERATOR$ - a variant of hash or multiply nested hash functions for acquiring the key used to encrypt subsequent communication