Department of Computer Science

Submitted in part fulfilment for the degree of
MSc in Cybersecurity.

Investigating the Security of p≡p's Trustwords

Aidan Fray

10 September 2019

Supervisor: Siamak F. Shahandashti

Word Count: 15401

Abstract

Many encrypted connections require the comparison of a fingerprint to protect against eavesdropping. A substantial amount of past work has aimed to propose fingerprint representations that work better with human limitations. "Trustwords" proposed by p≡p is an example of such a scheme, where the fingerprint is encoded as words in an attempt to improve usability.

This works main aim is to assess if Trustword's recommended minimum number of four words is sufficient. In order to achieve this goal, this work implements an attack on Trustwords and quantifies its effectiveness on more than 400 participants. A tool called GreenOnion was designed to assist in quantifying attack feasibility. GreenOnion improved substantially on a similar tool's ability to search for matches concurrently. Our findings show a substantial increase in attack success compared to related literature. We believe this increase is due to levering the design flaws in the Trustword scheme. Therefore, we believe a minimum of four Trustwords is insufficient to provide even a basic level of security.

Acknowledgements

I would like to thank my project supervisor, Siamak Shahandashti, for his continued support and enthusiasm throughout the project. Alongside this, I would like to thank the members of the *Human Verifiable Crypto* group Lee Livsey and David Norman for their ideas and discussion that helped create and cement ideas.

Contents

1 Introduction **1**

2 Background **3**
2.1 Project context 3
2.1.1 Public-key Cryptography and Key Exchange . . . 3
2.1.2 Encoding schemes 6
2.1.3 Similarity Metrics 8
2.2 Literature Review 11
2.2.1 Authentication ceremony performance 12
2.2.2 Encoding schemes 17
2.2.3 Attacks on encoding schemes 21
2.3 Overall Summary 23

3 Project Aims **25**
3.1 Research Questions 26

4 Design **27**
4.1 Overall attack design 27
4.2 Similarity metrics 28
4.2.1 Soundex 29
4.2.2 NYSIIS 29
4.2.3 Metaphone 29
4.2.4 Levenshtein Distance 29
4.2.5 Phonetic Vectors 30
4.3 Alternative Similarity Metrics 30
4.3.1 Match Rating Approach 30
4.3.2 Caverphone 30
4.4 Design of GreenOnion 31
4.5 Experiment Design 31
4.5.1 Metric performance 32
4.5.2 Trustword Attacks 36

5 Implementation **40**
5.1 GreenOnion 40
5.2 First Experiment 45
5.3 MainExperiment 47

6 Experimentation Results **50**
6.1 Scallion vs GreenOnion 50
6.2 Experiment 1 - Metric performance 52
6.3 Experiment 2 - Trustword attacks 56
6.4 Average number of near-collision keys for each metric . 59
6.5 Distribution of vulnerable keys 60
6.6 Generation of keys 61

7 Conclusions **63**
7.1 Research Question 1 63
7.2 Research Question 2 64
7.3 Research Question 3 64
7.4 Research Question 4 64
7.5 Further work 65
7.6 Final Remarks 66

A Randomly generated uncontrolled key **67**

B Computed Attack Keys **68**
B.1 NYSIIS - 0 Static 68
B.1.1 Controlled Key 68
B.1.2 Computed key 68
B.2 NYSIIS - 1 Static 69
B.2.1 Controlled Key 69
B.2.2 Computed key 69
B.3 NYSIIS - 2 Static 70
B.3.1 Controlled Key 70
B.3.2 Computed key 70

C Trustword Attack **71**

List of Figures

2.1 Photo depicting a MITM attack 4
2.2 Example business card with fingerprint 5
2.3 Random Art examples 7
2.4 Flag 7
2.5 T-Flag 7
2.6 Flag Ext 7
2.7 Variations of Flag 7
2.8 OpenSSH Visual Host Key 8
2.9 Unicorn 8
2.10 Vash 8
2.11 Examples of alternative graphical encoding schemes . . 8
2.12 Soundex mappings of letters to numbers 9
2.13 Trustword fingerprint verification 20
2.14 Re-mapping position 20
2.15 Best match obtained after a few minutes of hashing . . . 22

3.1 Most recent RFC security recommendation 25

4.1 Visualisation of the generation of near matches 28
4.2 Example experiment question 34
4.3 Exact experiment attention question 34
4.4 Failed verification of an incorrect checksum[38] 37

5.1 Bloom filter example 41
5.2 OpenCL Kernel 42
5.3 Exponent incrementation code 43
5.4 Main code used to generate match list 45
5.5 Example similar match function 46
5.6 Section of code from the First Experiment's Google App Script 47
5.7 Sequence diagram for the 'Lost Update' problem 48

6.1 Scallion's main print statement 51
6.2 Speed comparison between Scallion and GreenOnion . . 52
6.3 Soundex 55
6.4 Levenshtein 55
6.5 NYSIIS 55
6.6 Metaphone 55

6.7 Phonetic vector 55
6.8 Individual breakdown of results for each metric 55

C.1 Experiment UI 71

List of Tables

2.1 Examples for text based encodings 6
2.2 Phonemes to feature mapping table 11
2.3 Examples of vector addition 11
2.4 Average comparison time (seconds) for the encoding schemes 13
2.5 Overall accuracy of correct comparison for the encoding schemes assessed 14
2.6 Paper attribute comparison 16

4.1 The various phonetic encodings of the word "Travel" . . 31
4.2 Levenshtein's number of matches comparison 32
4.3 Phonetic vector number of matches comparison 33
4.4 Summary of attack requirements 38

6.1 Testing environment 50
6.2 Participant demographics 53
6.3 Average metric performance 53
6.4 Participant demographics 56
6.5 Internet Browser breakdown 57
6.6 Operating System breakdown 57
6.7 Success rates for simulated attacks 58
6.8 Each metric's average number of near-collision keys and average compute times 60
6.9 Number of vulnerable keys per metric 60
6.10 NYSIIS - OOOO 62
6.11 NYSIIS - XOOO 62
6.12 NYSIIS - XOOX 62

1 Introduction

The increasing use of public-key cryptography by instant messaging and secure email means ensuring confidentiality is an ever more important task.

One of the most significant risks to the security of the communication channel is a Man-in-the-middle (MiTM) attack. A MiTM attack involves an attacker impersonating one or both sides of a connection. MiTM attacks can entirely circumvent the encryption as it allows an attacker to read all of the encrypted data.

A countermeasure for the threat of MiTM attacks is the verification of each parties' fingerprint. A fingerprint is a small string of characters that is unique to each key and, thus, can be used to identify.

Fingerprints can come in several different encodings such as Hexadecimal, words, and even procedurally generated avatars. Previous research has shown that the average human can only hold around 7-digits (± 2) worth of data in their working memory[1]. Consequently, this makes the designing of user-friendly schemes a task of utmost importance.

Humans are commonly considered the most vulnerable part of any computer system. Solutions, therefore, have been proposed to remove this manual verification with examples such as PGP's web-of-trust[2] and Namecoin[3]. However, these suffer from user adoption due to perceived complexity. Manual verification is, consequently, left in a difficult position, due in part to proposed solutions needing to sacrifice either usability or security. Therefore, research into improving or creating secure and user-friendly fingerprint encodings remains an important task.

One proposed scheme claiming increased usability is Pretty Easy Privacy's (p≡p) "Trustwords". Trustwords is an implementation of word fingerprint mapping. This claimed increase in usability is achieved by the user comparing a reduced number of words compared to alternatives. This potential usability boost, however, comes at the cost of a much larger word list.

This report evaluates the security and feasibility of attacking Trustwords. The motivation for this is due in part to the shortage of justifica-

tion behind the size and features of the chosen wordlist.

The report is structured as follows:

- **Chapter 2** provides any required background knowledge alongside a review of all relevant literature.
- **Chapter 3** formally defines the aim of this report alongside the research questions the project aims to answer.
- **Chapter 4** documents the proposed designs required to answer the research questions proposed in Chapter 3.
- **Chapter 5** discusses the significant implementation details of the designs formalised in Chapter 4.
- **Chapter 6** presents and analyses all experimentation results of the project.
- **Chapter 7** evaluates and concludes the success of the project alongside a discussion into proposed further work. The research aims proposed earlier in Chapter 3 are used as evaluative metrics.

2 Background

The chapter explains aspects required to fully comprehend the project's achievements alongside an evaluation of current literature and the respective gaps.

The chapter contains the following sections:

- **Project context**
  Provides the user with the base knowledge required to understand the context and purpose of the following literature review.
- **Literature Review**
  A review of currently available literature on relevant topics. Recommendations and research gaps are identified and discussed.

2.1 Project context

To fully understand the literature review, it is necessary to introduce and explain the base knowledge before continuing.

2.1.1 Public-key Cryptography and Key Exchange

Asymmetric cryptography facilitates the secure encryption of messages in end-to-end encryption (E2EE), verification of digital signatures, and sharing of pre-communication secrets, among others.

The asymmetry stems from the use of "Public" and "Private" keys. The public-key is used to encrypt data that only the respective private-key can decrypt. Hence, this means that sharing the keys required to encrypt can be performed across insecure channels.

An E2EE connection uses the key types discussed previously to exchange messages between verified parties securely. However, this hinges on the verification of the initial parties, if one party impersonates another and receives sensitive communication, the secure encryption is completely circumvented. Therefore, the correct identification of parties is crucial to maintaining security. The exploitation of this is known as

a Man-in-the-middle (MiTM) attack. This attack is commonly bidirectional, and if executed correctly, there is often no discerning change to the user experience. Figure 2.1 contains a visual representation of this attack. In the diagram Eve (attacker) is impersonating Alice to Bob and Bob to Alice. Therefore, as both directions of communication are compromised, all traffic routes through the attacker Eve.

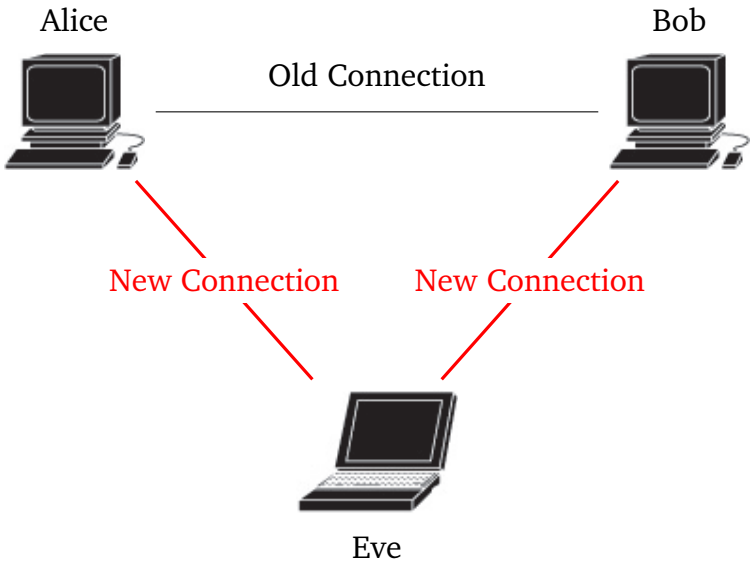

Figure 2.1: Photo depicting a MITM attack

Due to the assumption that the attacking party (Eve) does not have the same public-private key pair as either party (Alice or Bob), the unique aspects of the keys can be used to identify parties. This is achieved succinctly through the use of a fingerprint. A fingerprint is a string of alphanumerical characters that are unique to each key. Therefore the fingerprint of the two respective public keys are different, and, thus, can be used for identification.

The fingerprint of the key is generated by running the main key components through a secure one-way hash function such as SHA-256. This process produces a digest of a fixed length that can be used to compare keys. Therefore, the comparison of expected and actual fingerprints can be used to detect MiTM attacks.

Historically fingerprints have been represented as a hexadecimal string; whereon verification fingerprints are compared between two substrates, for example, a monitor screen and a business card. Figure 2.2 shows an example business card. A user then uses the fingerprint present on the card to compare to the one available on a device. This process has to be performed by each user and is known as the *"authentication ceremony."*

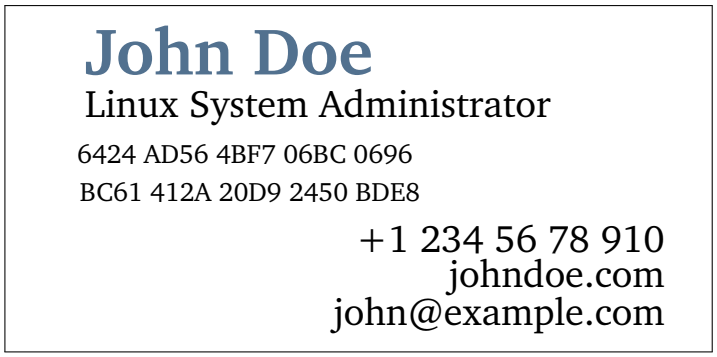

Figure 2.2: Example business card with fingerprint

System designers are, however, moving away from the structure of one fingerprint per identity and are now implementing a combined key. This change is due to the assumed increase in usability. Users now compare a single fingerprint together, instead of two sets separately. In the case of WhatsApp for example, the connection fingerprint of two parties keys is the first 30 bytes of an SHA-512[1] hash of each parties' identity key; these parts are then concatenated together to form a single fingerprint[4]. This process produces a unique key per communication pair.

Due to the manual comparison of the fingerprint, length and format are core design considerations. Prior research has shown that the average human can only hold around 7-digits worth of data in their working memory[1]. These findings show that comparing complete digests would be cumbersome and error-prone. For example, SHA-1 is 40 hex digits (160-bit) and, therefore, difficult to effectively compare. Therefore, if human interaction is required; there is a need for schemes that work effectively with consideration to human limitations.

[1] 5200 iterations

2.1.2 Encoding schemes

Text-based schemes

Encoding schemes are the physical method of encoding used to represent the fingerprint. The most common representation is alphanumerical with either **Hexadecimal** (0-9/A-F) or **Base32** (2-7/A-Z). These encoding schemes are the most popular due to their intuitive simplicity and lack of hardware requirements.

Fingerprints can also be encoded using natural language. The fingerprint can be chunked and mapped to a set of words. The same principle can be used to fill placeholders in a pre-defined sentence allowing the simple formation of syntactically correct English sentences. Moreover, other languages can be used such as Chinese, Japanese or Korean, to map fingerprint chunks to characters.

Hexadecimal	`6424 AD56 4BF7 06BC 0696` `BC61 412A 20D9 2450 BDE8`
Base32	`V623MNTZPL7VGU7A2CP7M6W4HSKP67IS`
Words	`telling realise tidy serve`
Sentences	`The basket ends your right cat` `on his linen.`

Table 2.1: Examples for text based encodings

Graphical Schemes

Graphical schemes are a potential alternative to text-based schemes. They often attempt to visualise small changes in the fingerprint to assist in alerting the user to small changes. This schemes should, therefore, be able to provide increased usability and security. This section explains the relevant schemes discussed later in this paper.

Random Art. Random Art engine[5] takes any length of input and processes it via SHA-1 to produce a digest that is then converted into an image. Figure 2.3 contains three examples from the Random Art Gallery[2].

Flag. Flag is a visual hash scheme proposed by **C. Ellison** and **S.**

[2] http://www.random-art.org/

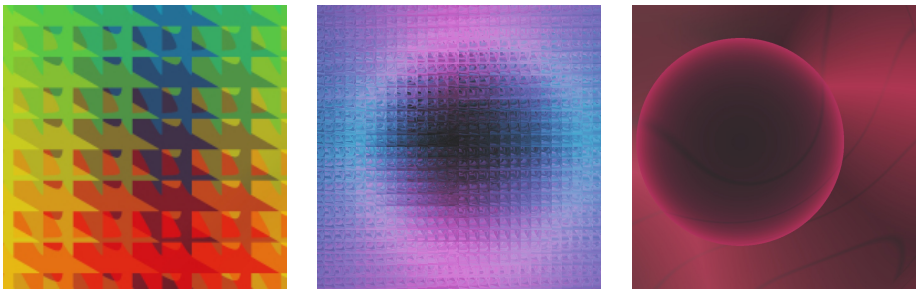

Figure 2.3: Random Art examples

Dohrmann[6]. Flag consists of four coloured strips with 2^n number of possible colours in each strip.

T-Flag. T-Flag is an attempt to improve on the previously mentioned Flag [7]. Improvements include twice the number of coloured blocks with a choice of eight colours. The authors claim the choice of colours are "colourblind proof".

Flag-Ext. Flag Extended is the proposed scheme assessed in the paper by **H. Hsiao *et al.***[8]. Flag-Ext aims, again, to improve on flag by reducing the number of blocks while adding 8 possible shapes.

Figure 2.7 contains an example of each version of Flag discussed.

Figure 2.4: Flag Figure 2.5: T-Flag Figure 2.6: Flag Ext

Figure 2.7: Variations of Flag

OpenSSH. OpenSSH Visual Host Key is an ASCII based graphical scheme used in MiTM detection when connecting to SSH servers. The image is generated by the movement of the central marker across a grid. Every 2-bits of the generated MD5 hash defines the diagonal direction of movement. For example, if the first 2-bits are `11`, the marker moves South East. The different characters are determined by the number of times the marker is present in that index, for example, if the marker is present on a square 4 times the character is an equals sign (=).

Unicorns[3]. Unicorns can be considered an example of an “Avatar” graphical scheme. The provided hash changes aspects of the image, such as background colour, horn length and colour of the Unicorn’s mane.

Vash[4]. Vash is similar to that of unicorns, where the hash determines specific characteristics of the image. Vash claims to have $\approx 5,438$ bits of entropy.

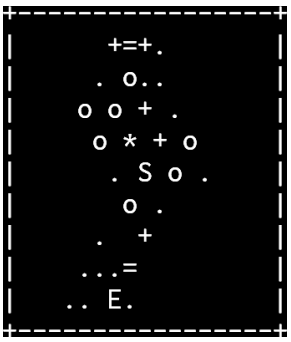

Figure 2.8: OpenSSH Visual Host Key

Figure 2.9: Unicorn

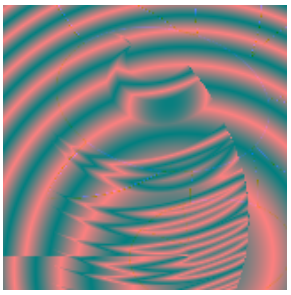

Figure 2.10: Vash

Figure 2.11: Examples of alternative graphical encoding schemes

Research in this area has taken these various encoding schemes and compared their fallibility to impersonated attacks. The main elements of comparison have been the “accuracy of attack detection" and “time to compare." In this paper, these are considered the metrics of “security” and “usability,” respectively.

2.1.3 Similarity Metrics

A similarity metric is an algorithm designed to determine if two words are phonetically a match. For example, the words "THEIR" and "THERE" is a match, whereas the words "DARK" and "PRINCIPLE" are not phonetically matching. This section provides a base explanation of important phonetic algorithms relevant in this project.

Soundex

One of the earliest examples of a phonetic algorithm is known as Soundex. It was initially designed for phonetically indexing names alongside

[3] https://unicornify.pictures/
[4] https://github.com/thevash/vash

detection of transposed letters in spelling mistakes.

Soundex produces a four-digit code for each word assessed. The first letter of the word is retained alongside the removal of all of `a`, `e`, `i`, `o`, `u`, `y`, `h` and `w`. The remaining letters are then mapped to numbers. These mappings are displayed in Figure 2.12.

```
b, f, p, v                1
c, g, j, k, q, s, x, z    2
d, t                      3
l                         4
m, n                      5
r                         6
```

Figure 2.12: Soundex mappings of letters to numbers

Due to the fixed length and limited digit set the initial concerns from this design is the limited number of combinations. There are a total of 5616 codes due to 26 initial letters and three digits of 6 values ($26 * 6^3$). The limited combinations result in matches of limited quality.

Further issues posed in [9] are discussed below and show the further deficiencies of Soundex in a dictionary word matching context.

1. **Dependency on the first letter:** Soundex cannot match words together if their first letters are different, meaning, for example, the words "KORBIN" and "CORBIN" are considered non-matching.
2. **Silent consonants.** Soundex does not have logic embedded to deal with silent consonants.
3. **Poor precision.** Due to the previously discussed point of a limited code space. [9] re-iterates this point but in the context of name matching where Soundex's poor performance was demonstrated. Soundex only gained and overall accuracy of 36.37% when matching names within a provided database.

NYSIIS

The New York State Identification and Intelligence System (NYSIIS) phonetic code was created for the phonetic matching of American names. The motivation for its inception was mostly due to the presence of Hispanic names in the American based databases (this was an aspect Soundex was known to have low accuracy with).

It also allows for variable-length codes and, thus, allows the applicability of the application to increase due to it not confronting the limited

code issue of Soundex. It was in use right up until the end of 1998 within various US Government departments.

Metaphone

Metaphone was invented by Lawrence Philips in 1990[10] in response to the deficiencies in Soundex. It improves on Soundex by including information around inconsistency and variation in English spelling in an attempt to create a more accurate phonetic representation.

Levenshtein Distance

Levenshtein distance is a string metric designed to measure the 'distance' between two strings. It is merely the number of single-character edits (insertions, deletions or substitutions) required to reach the other string.

An example distance between `trace` and `place` would be the substitutions of the first to letters, from `tr` to `pl`, meaning the two strings have a Levenshtein distance of 2.

Phonetic Vectors

Phonetic Vectors is a unique addition to the chosen set. Created by Allison Parrish in 2017[11], Phonetic vectors is as the name suggests the vectorisation of the phonetics of a word.

Phonetic features are used in this work as a way to compare the similarity of phonemes. Phonemes are the phonetic elements that construct a word. For example, the word "RING" translated into the phonemes `/R IH NG/`.

Extensive prior work has gone into producing models of features that map to phonemes [12][13][14]. Features, therefore, are an attempt at mapping the varying and inconsistent rules around the pronunciation of the English language. The vectors were created using lists phonemes from the CMU Pronouncing Dictionary and mapping them to possible features.

Table 2.2 contains the mappings used in [11] to create the phonetic feature lists. Using this with all 133,852 entries in version 0.7b of the CMU Pronouncing Dictionary, 949 unique properties were produced overall.

Phone	Features	Phone	Features	Phone	Features
AA	bck, low, unr, vwl	F	frc, lbd, vls	P	blb, stp, vls
AE	fnt, low, unr, vwl	G	stp, vcd, vel	R	alv, apr
AH	cnt, mid, unr, vwl	HH	apr, glt	S	alv, frc, vls
AO	bck, lmd, rnd, vwl	IH	fnt, smh, unr, vwl	SH	frc, pla, vls
AW	bck, cnt, low, rnd, smh, unr, vwl	IY	fnt, hgh, unr, vwl	T	alv, stp, vls
AY	cnt, fnt, low, smh, unr, vwl	JH	alv, frc, stp, vcd	TH	dnt, frc, vls
B	blb, stp, vcd	K	stp, vel, vls	UH	bck, rnd, smh, vwl
CH	alv, frc, stp, vls	L	alv, lat	UW	bck, hgh, rnd, vwl
D	alv, stp, vcd	M	blb, nas	V	frc, lbd, vcd
DH	dnt, frc, vcd	N	alv, nas	W	apr, lbv
EH	fnt, lmd, unr, vwl	NG	nas, vel	Y	apr, pal
ER	cnt, rzd, umd, vwl	OW	bck, rnd, smh, umd, vwl	Z	alv, frc, vcd
EY	fnt, lmd, smh, unr, vwl	OY	bck, fnt, lmd, rnd, smh, unr, vwl	ZH	frc, pla, vcd

Table 2.2: Phonemes to feature mapping table

The author then performed principal components analysis[5] on the unique properties to reduce them down to 50.

This metric allows for a unique set of actions to be performed on the phonetic output. Not only does this metric allow the user to measure *dissimilarity* (as opposed to the similar-or-not method of the alternatives) the continuous nature of the value allows mathematical operations to be performed on the output. An example shown in [11] was the addition of word vectors.

No	Operation	Result
1	$Vec(\texttt{sub}) + Vec(\texttt{marine})$	`submarine`
2	$Vec(\texttt{miss}) + Vec(\texttt{sieve})$	`missive`
3	$Vec(\texttt{fizz}) + Vec(\texttt{theology})$	`physiology`

Table 2.3: Examples of vector addition

For example, the addition of vectors can be seen in Table 2.3. This works for any mathematical operation with examples like multiplication allowing the 'tinting' of words with a phonetical theme.

2.2 Literature Review

This section discusses the current research around the proposed topic.

The organisation of the section is as follows:

- **Authentication ceremony performance**
  This section explains the current research on the performance of various encoding schemes and methods of comparison, alongside an assessment of the literature's experimental design.

[5] Details regarding this process are outside the scope of the project. Please, however, if interested please refer to this resource: http://setosa.io/ev/principal-component-analysis/

- **Encoding schemes**
  The following section then discusses research around the design of an encoding scheme. The focus of the section is the technical details regarding the creation and design of schemes.
- **Attacks on encoding schemes**
  The final section discusses literature that investigates the creation of actual attacks against encoding schemes.

2.2.1 Authentication ceremony performance

Encoding scheme performance

Results from the literature consistently show the effectiveness of language-based encodings such as Words or Sentences with accuracies ranging up from 94% [15][16][17]. In all cases, these were the best schemes from the sets assessed. The exception to this is the work performed by **H. Hsiao *et al.*** [8] in 2009 with Words achieving an abnormal accuracy of 63%.

Aside from textual representations were graphical schemes. As explained in Section 2.1.2, the primary graphical schemes assessed by the literature were: Random Art, Flag, T-Flag, Vash, OpenSSH Visual Host Key and Unicorns. These schemes had mixed accuracy with ranges as large as 50% - 94% in work by **Hsiao *et al.*** [8]. The only other paper assessing graphical representations was the work of **Tan *et al.*** [16], where they also achieved mixed results with accuracies ranging from 46% to 90%.

In terms of usability of graphical schemes, the literature concurred on their high usability. The comparison speed of these schemes were all among the quickest (See Table 2.4 for an overview of timings). Other work also indirectly agreed with graphical encodings having significantly quicker comparison times compared to non-graphical schemes [15], [17]. In terms of research into the performance of graphical schemes, the literature does not contain an extensive review with only two papers available. There is also no overlap in the schemes assessed with each paper reviewing a unique set. This area is, therefore, a promising candidate for further research.

One unique paper in this research area was the work by **M. Shirvanian, N. Saxena and J. J. George**[18] produced in the context of secure messaging pairing. This paper was unique for several reasons. First

[6] All values are quoted with the precision of their paper

Scheme[6]	H. Hsiao[8]	R. Kainda[17]	S. Dechand[15]	J. Tan[16]
Bit-Length	28 bits	20-40-bits	122-bits	128-bits
Hexadecimal			11.20	9
Numerical		6	10.60	9
Base32	3.51	6	10.20	
Words	4.63	7	8.70	7
Scentences		11	12.30	8
Chinese Symbols	5.01			
Japanese Symbols	5.07			
Korean Symbols	4.92			
Random Art	3.21			
Flag	4.28			
T-Flag	4.00			
Flag Ext.	4.02			
OpenSSH‘				5
Unicorns				3
Vash				3

Table 2.4: Average comparison time (seconds) for the encoding schemes

was the consideration for “remote-vs-proximity” pairing where this is the first consideration of this aspect found in the literature. There is room for further research to compare encoding schemes in the context of “remote-vs-proximity.” Another unique aspect was the end-to-end encryption context of the study.

The findings from the paper showed a high false-negative rate for all the schemes in a remote setting. This consideration is also missing from the literature. Alongside this, results for usability were lower in a remote setting for all the schemes. The author, however, comments on the expected nature of this result.
Images were identified as being the most secure method of authentication in the remote setting but voted as the method with the worst usability. This conclusion is highly inconsistent with all other work in this area. However, this could be due to the unique setting of remote verification, resulting in distinct results from user studies. Without further work in this area, it is difficult to validate these results conclusively.

Alongside this, it was shown in work by **H. Hsiao *et al.*** [8] that age and gender do not affect the accuracy of the scheme. However, younger participants were considerably faster. Furthermore, findings also showed that language comprehension helped in discerning small differences between schemes encoded in Chinese, Japanese, Korean, or English. Subsequently, knowledge of the language did not assist in differentiating more significant changes in the schemes as these had high accuracy

regardless. These were unique considerations. Further work could aim to corroborate these conclusions.

Scheme[7]	H. Hsiao[8]	R. Kainda[17]	S. Dechand[15]	J. Tan[16]	M. Shirvanian[18]
Hexadecimal			90%	79%	
Numerical		100.0%	94%	65%	97%
Base32	86%	86.7%	92%		
Words	63%	96.3%	94%	94%	
Sentences		100.0%	97%	94%	
Chinese Symbols	59%				
Japanese Symbols	57%				
Korean Symbols	54%				
Random Art	94%				
Flag	50%				
T-Flag	85%				
Flag Ext.	88%				
OpenSSH				90%	
Unicorns				46%	
Vash				88%	

Table 2.5: Overall accuracy of correct comparison for the encoding schemes assessed

Tables 2.4 & 2.5 contain the accuracy results of all papers assessed; this is to aid in visual comparison. Each paper used a different metric for measuring accuracy. Therefore, all results have been translated into "overall accuracy." For example, if a paper presented "security failures" or "attack success rate" of 5%, the overall accuracy of the scheme is 95%.

Methods of comparison performance

Method of comparison is the way a user compares the encoding schemes output. The most common example and the one used as default is "Compare-and-Confirm" (CaC). CaC, as the name suggests, is the *comparison* of two fingerprints on different devices or mediums that are then *confirmed*.

Aside from "Compare-and-Confirm" (CaC), there is "Compare-and-Select" (CaS) and "Compare-and-Enter" (CaE). CaS is the method where one device displays the fingerprint, and the other user is provided with several options. The user then has to choose the correct value from the list of candidates. If there is no match, the user must deny the connection attempt. The creation of CaS was due to concerns that CaC would be "too easy" for users leading to complacency and errors[19]. The design of CaE is for scenarios where both devices might not have

[7] All values have been rounded to the nearest whole percentage

a display, i.e., pairing between a phone and a keyboard. One device displays the checksum. This checksum is then entered into the other device. The first device then compares the entered string and checks for a match.

Research has assessed the performance of these schemes and how they affect the security of the authentication ceremony. The literature agrees on CaC being the best overall scheme to compare fingerprints [16][19] with CaS highlighted for its poor security and usability. CaE has had conflicting results. [19] discarded it after one round due to "poor usability." However, [16] considered it the best method overall for usability and security. These conflicting results, therefore, show polarisation in the results of CaE. However, this could be due to the different overall use-cases of the studies. Validation of results from either study would, therefore, be an area of additional work.

Experimentation methodology comparison

To further look into the validity of previously discussed results, it is necessary to asses how the respective studies reached their conclusions. Areas for consideration are scheme entropy, attacker strength, and participant demographics.

The starkest limitations of the literature are the range of participants and encoding entropy. The worst studies tested only 22-bits of entropy. This lack of entropy makes it challenging to compare results directly. One of the papers this most effects is the early work by **H. Hsiao *et al.*** [8] where their highest entropy is 28-bits. This level of entropy was inadequate even at the time of publication. There is an attempt by the authors' to address this issue in the later stages of the paper where they write: *"[...] increasing entropy is not a solution because it sacrifices usability and accuracy. With more entropy, representations will contain longer sequences of characters or more minute details which will lead to increased time and errors during comparisons."*. This statement is backed up with no empirical evidence, and the authors fail to consider how the low entropy would affect the overall security of the schemes.

Attacker strength is also another metric used to compare results concluded by each paper. Some papers failed to address their attacker strength consideration directly, but the overall strength has been inferred from the changes made to their schemes. For example, if they decided to change a single character in a 40-digit (160-bit) SHA-1 hex digest, they are indirectly stating that the attacker can control 39-digits (156-bit). To achieve this, the attacker would have to compute 2^{156} SHA-1 compressions to find a key match. In the literature, this element ranged from 2^{28} to around 2^{242}. However, this has some relation to the

size of the encoding schemes used. This substantial range makes it challenging to compare and confer results confidently. Further work could exclusively look into the effects that attacker strength has on the success of attacks. Moreover, all the papers assessed failed to fully consider the feasibility of attacks in terms of computer and storage requirements.

All of the studies considered demographical data when presenting their results. Their average ages were all around ~ 35 years old with the majority of participants educated with at least a bachelor degree. Alongside, the equal split between male and female participants.

One consideration of note is that made by **S. Dechand *et al.*** [15] where they briefly consider medical conditions such as ADHD and reading or visual disorders and the way they affect the comparison's effectiveness. The results highlight a slight reduction in overall accuracy; however, due to their small sample size, the results cannot be considered conclusive. This aspect is unique to all literature assessed. Further work would be required to produce conclusive results. Therefore, highlighting a gap in the literature.

One glaring issue with the demographical health of the study by **E. Uzun, K. Karvonen** and **N. Asokan**[19] is the use of two entirely different groups of participants. Not only were their demographics different, but they were from different countries (America and Finland). Different cultures contain inherent biases and assumptions. Changes were made pragmatically regarding the results from the first round of 40 participants. These were then re-assessed, alongside the direct comparison of results. This aspect casts vast doubts on the validity of the results with no consideration made by the authors to control external factors that may affect the performance of the method of comparison.

Hexadecimal and numerical schemes were not included as encoding schemes in work by **H. Hsiao *et al.*** [8] (some of the most widespread encoding schemes). The justification was the schemes similarities to Base32 alongside "well-known deficiencies". This point was provided with no further justification. Moreover, it is also inconsistent with available research, for example, in *"Empirical Study of Textual Key-Fingerprint Representations"*[15] it was shown numerical representations performed significantly better than that of Base32.

	H. Hsiao[8]	**R. Kainda**[17]	**S. Dechand**[15]	**J. Tan**[16]	**M. Shirvanian**[18]
Attacker Strength[8]	$\sim 2^{28}$	$\sim 2^{40}$	2^{80}	2^{60}	$\sim 2^{242}$
Entropy Range	22-28 bits	20-40 bits	122 bits	128 bits	160-256 bits
No° Participants	436	30	1001	661	25

Table 2.6: Paper attribute comparison

[8]Some attacker strengths are estimations due to the lack of quantification in some

Table 2.6 has been provided to compare the different aspects of the papers' parameters visually. It can be seen from the table the substantial ranges in participant size, entropy and attacker strength.

Conclusion

Overall, this review has identified several key areas suitable for further work. The first is the performance assessment of graphical encoding schemes. Recreation of pre-existing results from [8][16] is required to corroborate current conclusions and validate results. Another gap in the research is the consideration into the utilisation of encoding schemes in realistic conditions, i.e. "remote vs proximity." This topic was initially covered by [18], but their scope was limited.

Further work, therefore, could increase the scope and touch upon a large number of schemes in these settings. The final aspect for further work is the limited consideration into the feasibility of attacks on encoding schemes. All of the papers assessed simulated attacks and had minimal consideration for their execution. Therefore, further work could delve into the implementation and feasibility of such attacks.

2.2.2 Encoding schemes

Another area of research is investigations into the actual physical encodings of the hash digest. This section discusses the current research on the creation and security of actual encoding schemes. The technical and operation details are outside the scope of this literature review. Therefore, minimal attention is allocated to these areas.

Some of the oldest preliminary work into visual encoding schemes is by **A. Perrig** and **D. Song**[5] in the creation of their scheme "Random Art" in 1999. The motivation for creating such a scheme was the perceived flaws in the ways humans verify and compare written information. As mentioned in previous sections, visual encoding schemes have been shown to have mixed success, with low security being one of their most alarming flaws. This research laid the foundation for further work in analysing the security of visual encoding schemes.

Further research into the creation of unique visual hash schemes has been performed by **C. Ellison** and **S. Dohrmann**[6] (Flag), **Yue-Hsun Lin *et al.***[7] (T-Flag) and work by **M. Olembo *et al.***[20]. Each publication has provided a new way to represent a key fingerprint visually.

papers.

Alongside the academic literature, there are informally presented methods of visual fingerprints such as Unicorns[9] and Robots[10]. This list is by no means exhaustive but is used to depict the amount of research and work invested into graphical hash representations.

One paper of note is the preliminary work performed by **D Loss *et al.*** [21] in their *"An analysis of the OpenSSH fingerprint visualisation algorithm"* where they aimed to spur on further research with their initial findings into the security of the OpenSSH scheme. The authors claim that the use of the algorithm in OpenSSH is only heuristically justified, and a more formal proof is required.
The paper proposed several ways to generate similar fingerprints. The methods proposed were: Naive brute force, Graph Theory, and brute force of a full visual set. They were only able to produce only initial results and have proposed a large amount of potential further work. Since the paper's publication in 2009, there seems to have been no research building on the work of the authors.

Minimal research has also focused on elemental textual fingerprint representations and their security. Work by **A. Karole** and **N. Saxena**[22] looked into ways to improve the security of a textual representation. This research aim was to improve the secure device pairing process of comparing two numerical values. The devices used (Nokia 6030b; Mid-range devices at the time of publication) and the SAS (Short Authentication String) compared results in findings that are not directly applicable in a fingerprint comparison context.

A more specific subsection of textual fingerprints is the use of words and sentences to encode hash digests. Some of the first work in this area was by **Juola** and **Zimmermann** [23]. Their work aimed to produce a word list where phonetic distinctiveness was prioritised. Each word is mapped to a single byte. The unique aspect of the word list is the separation of "even" and "odd" words. Even byte indices are sampled from the even-list and odd indices from the odd-list. This technique effectively creates two sub-word lists. A genetic algorithm was used to maximise linguistic distance. The paper also includes a study on effective measures of "linguistic distances" of words and provided an in-depth discussion into these areas.

Overall the paper provides a foundation for formalising the creation of useful wordlists. A limitation is the lack of empirical data gathered on the performance. However, this was later evaluated in work by **S. Dechand *et al.*** [15] and shown to be an effective encoding scheme.

Other research of note is work by **M. Goodrich *et al.*** [24] called *Loud*

[9] https://unicornify.pictures/
[10] https://github.com/e1ven/Robohash

and Clear: Human-Verifiable Authentication Based on Audio. As the name suggests, the authors were researching ways to improve current methods of secure device pairing. The unique aspect of this work is the use of a Text-to-Speech system reading out syntactically correct English sentences. MadLibs[11] was the base for these sentences. MadLibs is where static placeholders are replaced with a member from a list of potential words.
This work is an extension to work performed by **Juola** and **Zimmermann**[23] as they aimed to emulate the techniques used in PGPfone. The paper's findings are limited by the lack of empirical data backing up claims made by the author on the performance of the system, and security is only theoretically assessed.

Aside from this research, there have been further informal implementations of fingerprint encodings – The first being by **Michael Rogers**[12]. Rogers' implementation is a program designed to map fingerprints to pseudo-random poems. This implementation was again, empirically evaluated by **S. Dechand *et al.*** [15]. Earlier work by **N. Haller** with the S/KEY[25] shows the implementation of a system designed to represent a hash as a series of six short words. However, this system is designed for a one-time-password purpose and only provides word mappings for basic human usability of the password and not within a fingerprint verification context. Therefore, the wordlist has not been designed with pronounceability in mind.

[11] https://en.wikipedia.org/wiki/Mad_Libs
[12] https://github.com/akwizgran/basic-english

Pretty Easy Privacy

A very recent implementation of a word list can be found in Pretty Easy Privacy (p≡p) implementation of TrustWords[13]. p≡p is a data encryption system that utilises PGP encryption to provide E2EE on all common channels of communication such as email or SMS. The embedded design principles state that above all the systems should be easy to install, use and understand.

p≡p deals with the threat of MiTM attacks by having users compare the respective key fingerprints encoded as a set of words. Figure 2.13 shows the p≡p Android implementation of Trustwords. The users then authenticate the words on an OOB (Out of Band) channel such as a phone call or in-person communication. If both users decide the words match, they accept or decline respectively.

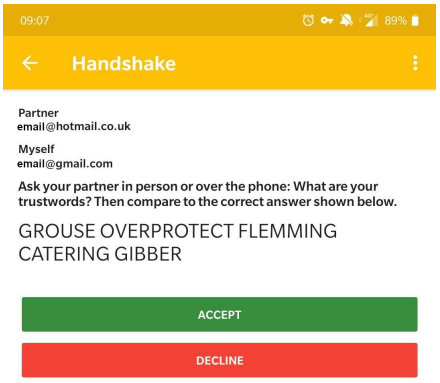

Figure 2.13: Trustword fingerprint verification

The unique aspect of TrustWords is its mapping of a single word to 16-bits. If compared to alternative literature, this is the highest number of bits-per-word seen. Full mappings (no duplication of words) would, therefore, require 2^{16} words in the dictionary. This number of words is arguably higher than most users' vocabulary. This deviation from the norm has not been currently backed up by research.

```
    [...]
52127 ZYGOTE
52128 ZYGOTIC
52129 ZYMURGY
52130 AACHEN
52131 AARDVARK
52132 AAREN
    [...]
```

Figure 2.14: Re-mapping position

Alongside the abnormally high number of words, the design choice to exclude slang and profanities from the English Trustword dictionary requires dual-mapping of a section of words. Approximately 13633/65536 (20.8%) of words are re-mapped in the dictionary, leaving 51903 unique words. The re-mapping is performed on an alphabetical loop. Figure 2.14 shows the position in the dictionary where this occurs.

[13] https://tools.ietf.org/html/draft-birk-pep-trustwords-03

Moreover, the primary RFC documentation remains in a draft stage and states: *"It is for further study, what minimal number of words (or entropy) should be required."*. These aspects highlight a substantial gap in the current literature.

Conclusion

In conclusion to this topic, the current research has primarily focused on the research and creation of visual representations. Research for textual fingerprints is fragmented and incomplete with work **Juola** and **Zimmermann** [23] and **M. Goodrich *et al.*** [24] providing meaningful research to build upon in terms of word and sentence based encodings. The fragmentation of this research leaves room for further work into this topic area. Alongside this, findings from the previous section's research show that human language based encodings provided the best usability and, therefore, should be a target for further research looking to improve upon their security and usability.

2.2.3 Attacks on encoding schemes

This area of research studies ways to physically execute attacks on fingerprint encoding schemes. This topic differs from previously examined work due to a central focus on the implementation details of an attack. Previous work discussed has only simulated the result of respective attacks. Research in this area is scant, with lots of research focusing on the security and prevention of Man-in-the-Middle (MITM) attacks.

Research in 2002 by **Konrad Rieck**[26] is one the first formalisation of attacks on fingerprint representations. The paper titled *"Fuzzy Fingerprints Attacking Vulnerabilities in the Human Brain"* aimed to look into ways users check hexadecimal encoded OpenSSH fingerprint representations. The author created an elegant way to 'weight' essential chunks of the digest. The bytes furthest to the right and left of the digests provided the highest weight. This technique provides a way to score digests and determine the best partial collisions found. For example, with the target fingerprint: `9F23` and partial match `9313` is given a score of 45% even though only two characters are matching.

The paper contains an implementation with a "1.2GHz CPU" being able to obtain 130,000 H/s (With MD5). In comparison to this, a mid-range Intel i5-3320M CPU can today obtain 111,700,000 MD5 H/s. This imbalance shows that the results obtained from the paper are significantly outdated. However, even with the low hash rate, the author was able to obtain some promising results. Figure 2.15 contains the best

example used.

```
TARGET: d6:b7:df:31:aa:55:d2:56:9b:32:71:61:24:08:44:87
MATCH:  d6:b7:8f:a6:fa:21:0c:0d:7d:0a:fb:9d:30:90:4a:87
```

Figure 2.15: Best match obtained after a few minutes of hashing

Overall the paper shows an interesting way to create partial fingerprint matches but is not quantified by any empirical evidence gathered on real-world users.

The only other relevant research on this topic is the work by **M Shirvanian and N. Saxena**[27] and their paper: *"Wiretapping via Mimicry: Short Voice Imitation Man-in-the-Middle Attacks on Crypto Phones"*. Further research in the area of "human voice impersonation" has received lots of attention [28][29][30]. This paper was chosen over other alternatives due to its specific use of encoding schemes in its evaluation.

In this paper, the authors develop a way to impersonate users when authenticating Short-authentication-Strings (SAS) in the pairing of Cryptophones. To achieve this impersonation they propose two methods: "Short voice re-ordering attack" where an arbitrary SAS string is recreated by re-ordering voice snippets obtained from eavesdropping a previous connection and "Short voice morphing attacks" whereby the use of previously eavesdropped audio snippets the attacker can morph their voice to match that of the victim. With these methods, they aimed to attack encodings of Numbers, PGP word list (previously discussed work by **Juola** and **Zimmermann** [23]) and MadLib (**M. Goodrich *et al.***[24] work also previously discussed). The effectiveness of these attacks was evaluated with a study involving 30 participants.

Results from the paper show the effectiveness of these methods. Compared to the baseline of the attacker's voice replacing the victim, performed with an $\sim$18% success rate. Morphing gained an overall success rate of 50.58% and re-ordering a very impressive 78.23% success rate and showing that these attacks provide an improvement on top of the naive implementation.

One of the most significant limitations addressed by the authors was the reduction in success rates as the size of the authentication string grew. The morphing and re-ordering attacks become increasingly ineffective as the user had more time to detect imperfections. The author does not quantify this aspect, and the extent of this degradation is never empirically discussed. Therefore, the results from this study are only practical and applicable in a SAS context.

Conclusion

Overall the literature for this subtopic remains sparse and incomplete. Further suggested work could look into the feasibility of generating partial collisions for all textual representations alongside quantified effectiveness on users. The work would aim to focus on the various physical methods used and their feasibility. This is one area the previous literature has failed to cover and has only theoretically quantified attacker strength without consideration for the actual real-world cost.

2.3 Overall Summary

In summary of the literature gaps discovered; encoding scheme performance requires further work focusing on all graphical schemes to corroborate results made by other work. Furthermore, there is a gap with the assessment of encoding schemes in the context of "remote-vs-proximity" first proposed by **M. Shirvanian, N. Saxena** and **J. J. George**[18] as it would arguably provide a better simulation of real-world scenarios.

In the context of human demographics and the way they alter the performance of the schemes; further research is required into how the fluency of languages affects authentication ceremony. This further work would be a continuation of the initial work by **H. Hsiao *et al.***[8] and could allow the creation of schemes that are better suited to specific languages. Alongside this, there is a research gap in the investigation of how mental impairments affect the performance of a scheme. This aspect is useful due to the number of potential users being affected. For example, 7% of the population have been identified as having dyslexic tendencies[31]. This fact means that with a UK population of around 63,000,000[14] 4,410,000 people could benefit from encoding schemes designed to work well with dyslexia. Dyslexia, however, is only one of many impairments that could affect a scheme's performance, thus, contributing to the substantial effect further work could have on this area.

Other possible large areas for consideration is the lack of direct consideration for attack strength when assessing the vulnerability of encoding schemes. Issues included the previously discussed wide range of attack strength (2^{28} - 2^{242}) alongside the lack of consideration at all from the majority of papers. Moreover, the complete lack of computation and storage complexity considerations means this is a prime area for further research as it would improve the applicability of results.

[14]From the 2011 Census collated by the Office for National Statistics

The final significant research gap identified is the lack of justification for the newly created Trustwords. Abnormal design choices (16-bit per word) and its recent creation alongside the shown effectiveness of language-based encodings, such as words makes this a promising area for further research.

3 Project Aims

This chapter defines the selected gaps and the subsequent research questions extracted.

The chosen project aims to asses the security of p≡p's minimum recommendation of four Trustwords (As stated in Figure 3.1). As discussed in the previous chapter, Trustwords aims to sacrifice security for increased usability. The encoding scheme has been designed to assist this by having the user compare a reduced set of words. Moreover, issues with the dictionary's design, such as the presence of homophones and dual-mapping of words show the potential for possible vulnerabilities.

As discussed in the previous chapter, the identified research gap regarding the minimal consideration of attack complexity would be a suitable addition for this project. This choice is due to it providing the ability to asses actual performance and, thus, the actual real-world strength of Trustwords while providing research to a scant area. Therefore, the security assessment is supplemented with complexity considerations alongside an actual implementation of the attack proposed.

```
"Short Trustword Mapping (S-TWM) requires a number of
Trustwords that MUST retain at least 64 bits of entropy.
Thus, S-TWM results into at least four Trustwords to be
compared by the user."
```

Figure 3.1: Most recent RFC security recommendation

By sampling the p≡p documentation, they have defined the use case of the Trustword handshake: *"A handshake is done by comparing the Trustwords between two users through a separate communication channel (e.g. in person or by phone)".*[1]. It can be assumed due to the increased globalisation of the planet that the most common handshake occurrence is over the phone, and, therefore, not in person. This is the assumed context of the handshake. This means, to create near-matches, the similarity of words is quantified phonetically instead of visually.

[1] https://www.pep.security/docs/general_information.html#handshake

3.1 Research Questions

With the previously discussed points in mind, the following research questions have been proposed.

1. What are the best performing schemes to quantify phonetic similarity?
2. What is the time and computation complexity required to generate a 'similar' keys for a targeted key pair?
3. What percentage of attacks successfully deceive a user?
4. Is the recommended minimum number of four Trustwords enough to provide a basic level of security?

4 Design

This chapter discusses an overview of the proposed designs for elements required to answer the research questions. Alternative solutions are explained when relevant alongside a justification of design choices.

4.1 Overall attack design

The attack on Trustwords involves generating "near-collision" keys. This attempts to provide the base to answer Research Questions 2 and 4.

Near-collision keys are keys composed of a set of words that are deemed a match by the phonetic similarity metric (See Section 2.1.3).

As each combined key in Trustwords is an exclusive-or of both sides' public-key (Equation 4.1 shows this process) the attack is designed to target a single pair of users and requires recomputation for every attack target. This aspect is considered when discussing the attack feasibility. Each pair is split into an "Uncontrolled" or "Controlled" key. Uncontrolled is the receiver of the communication, and, thus, the key cannot be altered. The Controlled key is the one we are attempting to impersonate. It is assumed that there is the ability to replace the Controlled key with the malicious option. These descriptions are the terminology used throughout the paper. However, the uncontrolled and controlled keys can be swapped around with the ability to compute both directions. Thus, resulting in the possibility to intercept both directions of communication. This, however, requires performing the attack separately for both directions.

$$KeyFingerprint_1 \oplus KeyFingerprint_2 = TrustwordsFingerprint \quad (4.1)$$

When attacking, a similarity metric is used to compute a list of possibilities for each position in the target fingerprint.

Figure 4.1 shows the process of generating a subsection of the combinations. A list of each words' matches is generated and combined to create the comprehensive list of near-collision keys.

CHOKE BLUSHING FRIGHTENING HAND
COKE
SMOKE

CHOKE
COKE BLUSHING FRIGHTENING HAND
SMOKE

CHOKE
COKE
SMOKE BLUSHING FRIGHTENING HAND

Figure 4.1: Visualisation of the generation of near matches

Completing these steps produces a list of fingerprints that can be inserted into a tool designed to hash a large number of keys and search for matches. This aspect of using an extensive list to search for keys massively reduces the complexity of the search.

In summary, the attack steps are:

1. Compute all possible matches using a similarity metric on all words in a dictionary (Only needs performing once).
2. Select a target and allocate "Uncontrolled" and "Controlled" key identification.
3. Calculate all permutations of near-collisions for the key pair and produce a list of near-collision key fingerprints.
4. Use a list of near-collision keys in the mass computation of keys to find near-collision keys.

4.2 Similarity metrics

One of the first requirements for the attack is quantifying "phonetic similarity." This section is looking to provide an answer for Research Question 1. There are many algorithms currently available to provide this functionality. This section describes and explains the reasons they were selected for assessment later in the project.

4.2.1 Soundex

Soundex is one of the most famous examples of a phonetic algorithm (See Section 2.1.3). This is due in part to its implementation into major database clients like MySQL[32], Oracle[33] and PostgreSQL[34].

Soundex, however, has had extensive work focused on its deficiencies alongside quantification of its performance under certain conditions. However, even with these previously discussed issues, its algorithmic simplicity and popularity allows it to remain relevant for assessment in this project.

4.2.2 NYSIIS

NYSIIS primary use case was the matching of common names present in a database (See Section 2.1.3). However, due to it having embedded rules to handle word phonetics, it would again be applicable in this application.

Due to NYSIIS's aforementioned use in a professional setting alongside its high occurrence in literature, it was deemed suitable as one of the selected similarity metrics.

4.2.3 Metaphone

Metaphone, as discussed in Section 2.1.3, is another famous phonetic algorithm. Metaphone is arguably on the same level of ubiquity as that of Soundex with it finding itself implemented in languages such as PHP[35]. Metaphone was chosen due to similar reasons to that of Soundex and NYSIIS.

Further work would involve the implementation of newer versions of Metaphone. Double Metaphone (2000) and Metaphone 3 (2009) that claim to improve over the original version.

4.2.4 Levenshtein Distance

As explained in Section 2.1.3, Levenshtein Distance is the number of whole character edits. Levenshtein distance is not technically designed as a phonetic algorithm, but due to similar-sounding words often being spelt in similar ways[36], the use-case of this algorithm remains relevant. Alongside this, Levenshtein is also the most algorithmically simplistic

of all the considered options. These aspects, therefore, contribute to Levenshtein's inclusion in the chosen set.

4.2.5 Phonetic Vectors

As discussed in Section 2.1.3, the metrics ability to perform operations and measure dissimilarity allows for a plethora of applications. One possible application of note would be the creation of a wordlist where the phonetic difference (vector distance) is maximised. This application, if the vector mapping is an accurate representation, could allow the creation of a phonetically distinctive wordlist. Therefore, these unique aspects contribute to the inclusion of this metric in the set.

4.3 Alternative Similarity Metrics

4.3.1 Match Rating Approach

Matching Rating Approach (MRA) is another algorithm designed to match names within a database; therefore, its operation can be grouped with that of NYSIIS and Soundex. MRA was discarded as an option. This removal of this metric is due to the lack of known utilisation in any substantial real-world use-cases alongside its similarity to more established algorithms like Soundex and NYSIIS.

Further work could compare this algorithm to similar alternatives in phonetically matching of words to quantify performance. This further work is discussed in the evaluative sections of the report.

4.3.2 Caverphone

Another notable alternative solution is that of Caverphone that was designed in New Zealand. Caverphone, as is the case with the vast majority of phonetic algorithms, was designed for use with name matching. Caverphone was not chosen to similar reasons to that of MRA alongside its optimisation for the New Zealand dialect. Therefore, the low level of utilisation alongside the unique design features contributed to its exclusion. However, this has not been assessed empirically and thus would be a candidate for further work.

Metric	Output
Soundex	T614
NYSIIS	TRAVAL
Metaphone	TRFL
Caverphone	TRF111
MRA	TRVL

Table 4.1: The various phonetic encodings of the word "Travel"

4.4 Design of GreenOnion

In order to answer Research Questions 2 fully, it is necessary to implement a tool to generate working keys.

The inspiration for the design of this tool was taken from a tool called Scallion[1]. Scallion was designed by Richard Klafter and Eric Swanson and was used to demonstrate that 32-bit PGP key IDs were insufficient. To keep with the onion-based theme, the proposed tool is called 'GreenOnion' and is a re-write of Scallion in C++. This language was chosen due to the well-understood efficiency benefits. The proposed tool differs from Scallion, most notably in its ability to concurrently search for a large number of keys, GreenOnion improves on this substantially. More implementation and experimental details are discussed in later chapters.

The tool should take two keys as parameters (Uncontrolled/Controlled) and a chosen similarity metric and produce a list of target key fingerprints. This list is then used as a search criterion when searching for keys. To utilise the parallel nature of the GPU to compute the hash of a large number of keys, the tool utilises a GPGPU (General-purpose computing on graphics processing units) framework. The chosen framework was OpenCL due to its support for the chosen language (C++) and platform (Linux). OpenCL allows the creation of code chunks referred to as "kernels" to be executed concurrently, this provides a massive speed increase compared to the sequential nature of the CPU. Technical details are explained further in Chapter 5.

4.5 Experiment Design

To answer some of the research questions, empirical evidence is required. In this section, the designs and considerations of the chosen experiments are discussed.

[1] https://github.com/lachesis/scallion

4.5.1 Metric performance

This section aims to assist in answering Research Question 1. To reduce the number of metrics to assess in the later rounds, the performance of similarity metrics required comparison. The thinning out of metrics is required due to limited resources. Therefore, possible further work could involve repeating this work with a much more varied selection of metrics. As explained in a previous section (See Section 4.2) the chosen metrics up for assessment are Soundex, Metaphone, NYSIIS, Levenshtein and Phonetic vectors.

The design for the experiment involves assessing the quality of the metrics matches by having participants rate them on a scale of 1 (Very different sound) to 5 (Very similar sound).

Matches

A match for each code based metric (Soundex, Metaphone and NYSIIS) occurs when the codes are identical. However, for the other cases where the difference is variable other ways of defining a match are required.

In the case of Levenshtein values of similarity are discrete due to it being the number of single-character edits. Therefore, this requires less deliberation. Match size was used as a way to decide on a cut-off point. Match size is the number of matches determined by the metric on the English version of the Trustword dictionary (51903 unique words).

Metric	Matches
Soundex	1,527,554
Metaphone	412,916
NYSIIS	188,474
Levenshtein (Distance 1) [L1]	**97,730**
Levenshtein (Distance 2) [L2]	1,070,656

Table 4.2: Levenshtein's number of matches comparison

As can be seen from Table 4.2, the number of matches for L2 is more significant than all metrics excluding Soundex. As the issues with Soundex have been discussed in previous sections (See Section 2.1.3), it can be excluded as an abnormality in this context. Therefore, due to the excessively high value for a L2, L1 was chosen as the Levenshtein cap for defining a match.

Match size was also used to define the threshold for the phonetic vectors. The decision is less apparent than that of Levenshtein due to

the continuous nature of the distance between two vectors. However, the complexity was reduced by limiting it to increments of 0.5.

Metric	Matches
Soundex	1,527,554
Metaphone	412,916
NYSIIS	188,474
Levenshtein	97,730
Phonetic Vector (Distance 3.0)	14,550
Phonetic Vector (Distance 4.0)	73,962
Phonetic Vector (Distance 4.5)	**216,156**
Phonetic Vector (Distance 5.0)	685,516

Table 4.3: Phonetic vector number of matches comparison

Table 4.3 contains the match size comparison for Phonetic vectors distance. Distance 4.5 was chosen as a suitable size due to the total matches fitting well within the other metrics. Distance 4.0 was the other alternative; this was discarded due to its small size. Any metric with a number of matches below 100,000 results in an inadequate number of possible near-collision keys later in the process, a lower number of matches may imply better quality, but a balance is required between computational cost and attack quality. This is discussed in more detail in later chapters.

Comparisons

Each comparison contains a match-pair. These matches are generated by running the similarity metric over all the pairings in the Trustword dictionary; if the pair meets the criteria discussed in the previous section, it is determined a match. Potential matches are then sampled from their respective lists.

Each participant is then asked to rate the similarity of a match on a scale of 1 (*'Very different sound'*) and 5 (*'Very similar sound'*). Figure 4.2 shows an example match and the connected scale. Users are provided with five of these comparisons per similarity metric.

The experiment randomises the order of these comparisons per session alongside a complete refreshing of matches once per submission. This design makes sure selections from the samples are fair and consistent as each user has a different selection of matches.

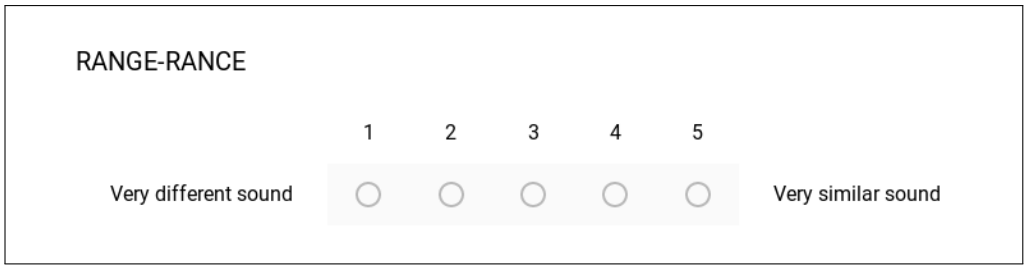

Figure 4.2: Example experiment question

Quality control

As the study was being outsourced to Amazon's Mechanical Turk, the requirement to check the quality of results is essential. Therefore, a couple of additions were provided to check a result's validity.

The first was the addition of five "Random matches" questions. These are matches consisting of two random words selected from the dictionary. Therefore, the expected average rating of these words should be close to one. This design allows for a simple check for valid results. If their average rating for Random matches is too high, their results are discarded.

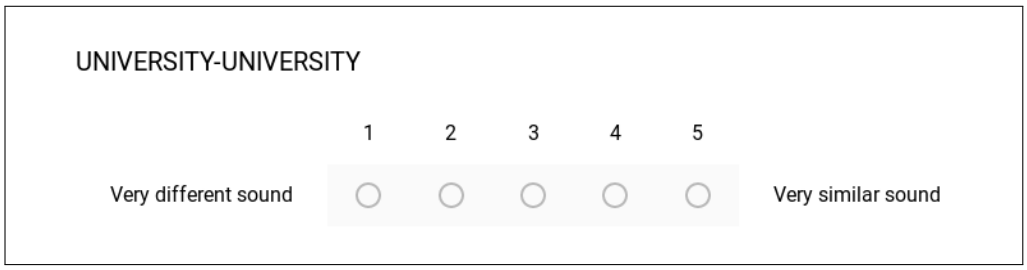

Figure 4.3: Exact experiment attention question

Alongside this, was the addition of two questions comparing precisely the same word. As both words are the same, the result should always be a full 5/5 rating, any results without a full rating are discarded. Figure 4.3 contains one example of the attention question used to filter inaccurate results.

The final check to ensure the validity of results is a check for native English fluency. Having non-native English speakers complete the experiment can introduce inconsistencies into the data and, therefore, needs to remain controlled. To achieve this, a preliminary MTurk study has been created to ask users for their perceived English fluency on a scale of 1 (Basic Understanding) to 5 (Native). Workers with an answer of 5

are provided with a Qualification[2] that tags them as defining themselves as native speakers. This qualification is then added as a prerequisite when running the experiment. This prerequisite ensures only fully native speakers are being assessed. The possibility of the worker falsely stating their level of fluency has also been considered. However, as the worker does not know the purpose of this initial study, thereby providing inaccurate results has no real valid motive for the worker.

Considerations

The first consideration is the way the words are compared. Due to the assumed context of authentication being phone-based, the comparison of words is a mixture of auditory and visual. The user hears their authentication partner read their set of words (Auditory) and compare to their set of words (Visual). This experiment asks users to compare the sound of two words that are visually displayed to them. This visual aspect is the first point of contact between the participant and the question, followed by a mental comparison of the phonetics of the word. Therefore, this initial visual contact has the potential to bias the results of the study. Levenshtein has the potential to be most affected by this due to the matches never being more than one character different, therefore, resulting in very similar-looking words. The alternative method of running this experiment is to get users to compare audio samples of words; this forces the user to compare just the phonetics of the words and not be influenced by the visual aspects of the pair. However, this adds a substantial cost to the operation of the study. Due to the aforementioned lack of resources, the chosen design was decided as most feasible as it is a balance between cost and accuracy. Further work, therefore, could improve by producing more conclusive results on the performance of the metrics by implementing the more accurate but costly alternative.

Another consideration is the demographics of people accessible on Mechanical Turk. A comprehensive and still active survey run by **D. Difallah, E. Filatova** and **P. Ipeirotis**[37] shows that MTurk workers are younger and have a more substantial household income than that of the US population. This deviation has to be taken into consideration when interpreting the results because it has to potential to introduce a bias.

[2] https://blog.mturk.com/tutorial-understanding-requirements-and-qualifications-99a26069fba2

4.5.2 Trustword Attacks

This experiment is designed to be a simulation of the attack proposed in Section 4.1 in an attempt to answer Research Question 3. The experiment tests the simulated matches of three similarity metrics decided by the previous experiment (See Section 4.2). The overall aim of the experiment is to quantify the success rate of the attack on live participants. The user is presented with the same design as the p≡p Android application (See Section 2.2.2).

Appendix C shows the design of the experiments front end[3]. As can be seen by comparing it to Figure 2.13, it has been designed to be as close as possible to the actual application. The blue button is clicked to simulate the authentication ceremony over the phone. A text-to-speech system then reads a set of words and the user should accept if they match and decline if they don't.

Design

Due to the scarcity of attacks in a real-world setting, users are often complacent about their occurrence. Therefore, in this experiment, attacks only occur 30% of the time. This occurrence is a much higher value than in a real-world setting but is a balance between resources and realistic design. If the chosen attack rate is too low, many more trials are required to collate a useful number of attack instances, thus, requiring more resources, but with more realism. If, however, the attack rate is too high, less total rounds are required to obtain the target number of attack trails. However, the user's complacency is lost as an attack is expected. This loss of complacency is undesirable as it does not realistically reflect real life. Furthermore, the first 5 trials of a new experiment are always benign. This inclusion is to ensure users are consistently lulled into complacency.

Certain keys have higher levels of potential near-collision keys than others due to how certain words are deemed similar by the similarity metric. Therefore, this presents the possibility to have keys with a deficient number of possible near-collision keys. The distribution of these keys is explained further in Chapter 6. Therefore, if random sets of words are chosen with no restraints, there are attacks presented to the user that in a real setting would be infeasible as they are close to a 2^{64} attack (4 Words * 16-bits). Therefore, the experiment is designed to sample from a list of "vulnerable" keys. These are keys that have a number of potential near-collision keys that allow computation in a specified timeframe.

[3]A live version can also be viewed at the url: https://afray.pythonanywhere.com/

Furthermore, certain levels of attacks are also simulated. Below is the list of attacks considered in this experiment:

- (OOOO) All words in the set can vary
- (**X**OOO) All but the first word can vary
- (**X**OO**X**) All but the first and last word can vary

Where :

X: is a static word

O: is a changeable word

The start and ends were chosen as the highest priority words to keep static. This design choice is due to research highlighting the common habit of users only to check the start and end of a checksum. [38] shows this by measuring user eye movement and displaying it as a heat map.

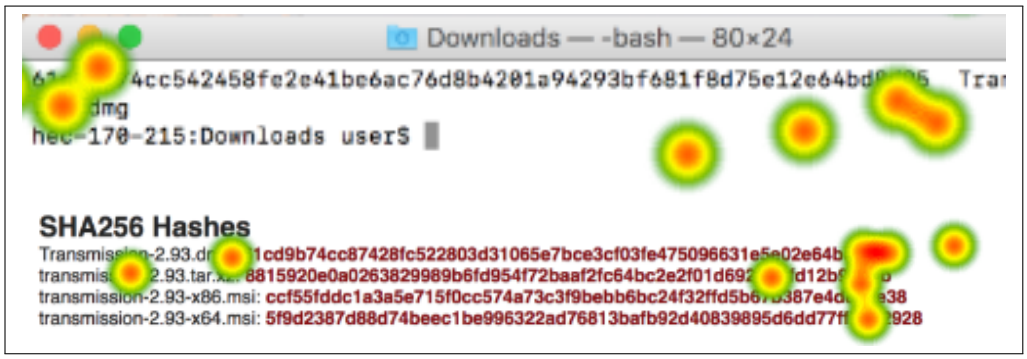

Figure 4.4: Failed verification of an incorrect checksum[38]

Figure 4.4 shows the results of an eye-tracking experiment where the user failed to detect a mismatching digest. As can be seen, the users never compared the centre of the digest.

However, this may not be applicable in this context due to the user being linearly fed an audio stream that they cannot skip to the start or end. We believe, however, that attention also plays a part in defining the most important areas of comparison. The hypothesis is that the users' attention is lower in the middle sections of comparison, where it peaks at the start and end. All of these aspects require empirical evidence to conclude. Therefore, this requires consideration when assessing the results of the experiment.

These attacks are ascending in complexity. The timeframe for computing a near-collision was decided as one week with attack strengths of 7 GPU-days *(1 x GPU)*, 70 GPU-days *(10 x GPUs)* and 700 GPU-days *(100 x GPUs)* on a mid-range GPU (AMD RX 480). Due to the long lifetime of

PGP keys, 7 days was chosen as a reasonable length of the attack. The tool requires no user interaction once started. Therefore, the tool can be left for an extended period to compute potential keys. This timeframe is also a maximum time, as these attacks also include keys that have a shorter average computation time.

GPU-days	No of near-collision keys	Attack Type
7	15250	○○○○
70	1525	**X**○○○
700	152	**X**○○**X**

Table 4.4: Summary of attack requirements

Table 4.4 contains a summary of the different level of attacks and their respective restrictions. More potential near-collision keys make the attack search quicker. Any key that exceeds the defined numbers of potential near-collision key for attack level is deemed as vulnerable. As discussed in Section 4.1, the near-collision keys are close keys produced from the similarity-metric's matches. A list of these vulnerable keys is sampled from when an attack is simulated. The percentage of vulnerable keys is explored in Chapter 6.

Quality control

Like the previous experiment, participants are sourced from MTurk. Therefore, again, quality control is an aspect that requires consideration. As with the previous study, initially screened native speakers are used. Alongside this, two metrics are used to detect invalid results: audio-button-clicks and the overall-time.

Audio-buttons-clicks is the number of times the blue "*Authenticate with partner over the phone*" button in Figure C.1 is clicked. If there are rounds without button clicks, this is a sign of non-attentiveness. The design choice was made to allow non-clicks as an attention check. This design provides a way to detect and discard click-throughs[4].

Overall-time is as the name suggest a recording of the time taken to complete the entire experiment. If users complete the experiment in an abnormally small amount of time, it can be used as another detection for invalid responses. A benchmark for the time is set by completing the experiment as fast as possible. Anything quicker was discarded as anything faster risks invalid and rushed results.

[4]Workers that aim to complete the task as fast as possible, with no regard for the quality of responses

Considerations

As with the previous experiment, sampling from MTurk's demographic results in a collection of participants that are not reflective of the general population. Due to the attack context of this experiment, it can be assumed that the youth of the participants contributes to less successful attacks. Therefore, the results generated by this experiment can be loosely assumed as a best-case scenario for the scheme. We predict a high attack success rate if assessed over the general population.

5 Implementation

This chapter discusses the implementation details of the project. Instead of explaining all the technical achievements in detail, the most interesting ones are discussed alongside the respective challenges encountered.

5.1 GreenOnion

General design

This section briefly discusses the overall technical implementation of GreenOnion alongside a more in-depth discussion into the unique aspect of the tool.

GreenOnion is written in C++. This language was chosen due to its efficiency bonuses over other alternative languages. The tool begins by generating a 2048-bit RSA key through GPG[1]. This key is then used to create a hash of all but the final 512-bit block of the key. The final block of the key is left un-hashed due to the presence of the exponent (e). This exponent is then incremented to create a new unique key. This technique allows the accelerated generation of new keys due to the reduction in expensive large prime generation. Entropy starvation is an example of a common issue encountered when generating a large number of RSA keys. This process produces valid keys, but with abnormally large exponents, this was deemed suitable due to the short term use-case for these keys. Three bytes are used to represent the exponent giving the potential to create $2^{24} - 1$ extra keys for a single expensive key generation.

The intermediate hash is then loaded on the GPU via an OpenCL kernel. This kernel is designed to increment the exponent, hash the final block, obtain the final fingerprint and check if the fingerprint is present in the provided data structure.

[1] GNU Privacy Guard

Bloom filter

Mass checking of fingerprints is the tool's central ability as it allows for millions of keys to be checked with a minimal amount of overhead. This functionality is achieved through the use of a bloom filter. A bloom filter is a probabilistic data structure that allows efficient checking for the presence of an element in a set. It is effectively a vast array of booleans that state whether an element is present. A pre-defined hashing algorithm decides the index in the array. This process is repeated several times with a set number of hashing algorithms (k) to populate the array of length (m). To check the presence of an element in the set means hashing the target value and checking the indices returned. This data structure, therefore, has a complexity of $O(k)$ regardless of the number of elements in the set. This is the only data structure that provides this characteristic. This benefit comes at a cost. Due to the use of hashing algorithms, there is the possibility for collisions and, thus, the possibility of false-positives. The data structure, however, does not produce any false-negatives.

These qualities, therefore, makes it fully suited to this use-case as any fingerprints tagged as "possibly" being present in the bloom filter can be followed up with a more expensive hash-table check to determine their actual presence. Therefore, if the levels of false-positives are controlled (by altering k and m), the tool can search through a vast number of potential keys without any decrease in speed.

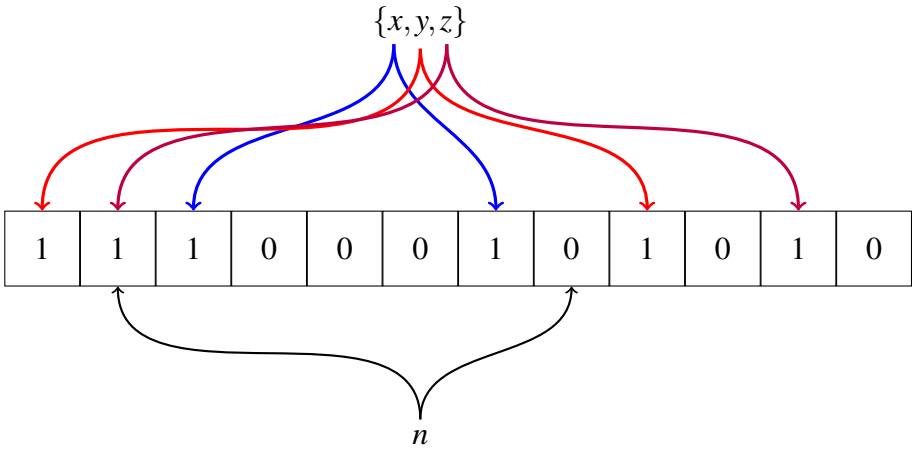

Figure 5.1: Bloom filter example

Figure 5.1 shows the operation of a simple bloom filter. As can be seen, the element n does not exist in the set due to both array indices not being set. The performance boost when comparing large numbers of keys is substantial when compared to a similar tool Scallion (See Section 4.4.) The results from the comparison are quantified in Chapter 6.

OpenCL

In order to process a substantial number of keys in a concurrent fashion, OpenCL was required. Figure 5.2 contains the main sections of the code uploaded to the GPU.

```
__kernel void key_hash(__global uint* finalBlock,
                       __global uint* currentDigest,
                       __global bool* bitVector,
                       __global long* bitVectorSize,
                       __global uint* outResult)
{
  uint W[16];
  uint H[5];

  W = finalBlock;
  H = currentDigest;

  // <EXPONENT_INCREMENTING_CODE>

  // Hashes the final block
  sha1_block(W, H)

  // Checks against the bloom filter
  bool match = true;
  for(int n = 0; n < 2; n++)
  {
    uint l = MurmurHash3_x86_32(H[0], n) %
             bitVectorSize[0];

    uint r = MurmurHash3_x86_32(H[1], n) %
             bitVectorSize[0];

    if(!bitVector[l] || !bitVector[r])
    {
      match = false;
      break;
    }
  }

  // Returns parts needed to recreate the key
  if(match)
  {
    outResult[0] = 0x12345678;
    outResult[1] = newExponent;
  }
}
```

Figure 5.2: OpenCL Kernel

The `<EXPONENT_INCREMENTING_CODE>` is explained in the next section. After the exponent is incremented, the code produces the final digest by calling `sha1_block(W, H)`. The 32-bit left and right sections of the digest are hashed using the MurmurHash algorithm. MurmurHash does not exhibit features suitable for secure hashing as it can be trivially reversed. Its use-cases include hash-based lookups due to its speed and its distributive properties, making it suitable to work well with a bloom

filter. The hashed values are then checked against the bloom filter array. If there is a possible match, the identification value (`0x12345678`) is placed in the first byte of the `outResult` alongside the exponent used to produce the possibly matching PGP key. The minimal amount of data returned to the CPU via the `outResult` variable is due to efficiency reasons. Use of this memory is computationally expensive; therefore, at every opportunity, writing or reading this memory has been avoided.

```
1  //The exponent is split across two words
2  // i.e
3  //
4  // W[3] = XX EE EE EE
5  // W[4] = EE XX XX XX
6  //
7  int newExponent = get_global_id(0) + \
8                    ORIGINAL_EXPONENT_VALUE;
9
10 // Moves the value to the right i.e. 0x01ffff -> 0x0001ff
11 // Diving by "0x100" moves the hex value 3 digits
12 // to the right
13 int exponentMaskLeft = ((int)(ORIGINAL_EXPONENT_VALUE / 0
       x100)) ^ ((int)(newExponent / 0x100));
14
15 W[3] = W[3] ^ exponentMaskLeft;
16
17 // Grabs the last byte of each exponent to apply to
18 // the next word
19 int lastByte = (int)(ORIGINAL_EXPONENT_VALUE / 0x100) *
20     0x100 ^ ORIGINAL_EXPONENT_VALUE;
21
22 int lastByteNew = (int)(newExponent / 0x100) * 0x100 ^
       newExponent;
23
24 // Moves it into position as the MSB of a 32-bit integer
25 // i.e 0xff -> 0xff000000
26 int exponenetMaskRight = (lastByte * 0x1000000) ^
27     (lastByteNew * 0x1000000);
28
29 W[4] = W[4] ^ exponenetMaskRight;
```

Figure 5.3: Exponent incrementation code

Figure 5.3 contains the code used to increment the exponent. Due to the exponent being split across two words alongside the limited functionality of OpenCL, bitwise operations are used to increment the exponent. The code utilises the fact that anything XORed with itself becomes zero alongside the fact that XORing a value a with zero provides the result equal to a. The code manipulates the position of values by multiplication or division of base-16 values; these shift the hex value

either left or right, respectively. Once it is in position the "Mask" is created by XORing with the original value and the new intended value, this stores the changes required to reach the required values, this then applied to each of the words. The `get_global_id` command takes the ID from the currently running thread; this allows a different value to be used for each exponent concurrently.

5.2 First Experiment

As discussed in Section 4.5.1, a list of matches were generated for each metric. Figure 5.4 shows the algorithm designed to compute these lists. It loads the entire Trustword dictionary and works through all combinations of words.

```
1  void find_similar_words(vector<string> words)
2  {
3      vector<string> sim_words;
4
5      for (string word_y : words)
6      {
7          sim_words.push_back(word_y);
8
9          for (string word_x : words)
10         {
11             bool add_word = false;
12
13             else
14             {
15                 if (LEVENSHTEIN) add_word =
16                     leven_similar(word_y, word_x);
17
18                 else if (METAPHONE) add_word =
19                     meta_similar(word_y, word_x);
20
21                 else if (NYSIIS) add_word =
22                     nysiis_similar(word_y, word_x);
23             }
24
25             if (add_word)
26             {
27                 sim_words.push_back(word_x);
28                 add_word = false;
29             }
30         }
31     }
32 }
```

Figure 5.4: Main code used to generate match list

Figure 5.5 further shows the code used in the `leven_similar`. The Levenshtein difference is computed and compared to the tolerance discussed in Section 4.5.1. If this function returns true the match is added to the list.

```
1 bool leven_similar(string word1, string word2)
2 {
3   int diff = lev_distance(word1, word2);
4   return diff <= DIFFERENCE_TOLERANCE;
5 }
```

Figure 5.5: Example similar match function

In order to assess and collect data from users for the first experiment (See Section 4.5.1), Google Forms[2] was used to host and design the questionnaire. It provided the functionality to present questions easily and record responses. However, as the design requirement of refreshing questions after each submission was not part of the main applications functionality, an extra solution was required. This problem was solved by Google's App Script[3] service that allows a user to interact with the Form programmatically. This addition allowed the re-sampling of matches each time a response is submitted.

Figure 5.6 contains an interesting snippet from the script used in the experiment. Each time a user submitted their response, the `updateForm()` endpoint was called, that subsequently updated the questions for the next user. This technique aimed to keep the samplings fair as they are equally likely to appear when attacking the system.

[2]https://www.google.com/forms
[3]https://developers.google.com/apps-script/

```
var ALGOS = ["Soundex", "Metaphone", ...];
var ALGO_SIZES = [763777, 412916, ...];

var QUESTION_PER_ALGO = 5;

function updateForm(ss, form) {

  // Obtains a list of the items present on the form
  formItems = form.getItems();

  var algo_index = 0;
  var scale_index = 0;
  for (var i = 0; i < formItems.length; i++)
  {
    if (formItems[i].getType() == "SCALE")
    {
      var rnd_index = Math.floor(Math.random() *
      ALGO_SIZES[algo_index]) + 1;

      // Obtains the Google Sheets object
      var sheetname = ALGOS[algo_index];
      var sheet = ss.getSheetByName(sheetname);
      var range = sheet.getRange(rnd_index, 1)
      var values = range.getValues();

      // Re-samples from the connected Google Sheets
      _updateScaleTitle(formItems[i], values[0]);

      scale_index++;

      // Moves on to the next metric when a group size
      // has been completed
      if (scale_index != 0)
      {
        if (scale_index % QUESTION_PER_ALGO == 0) {
            algo_index++;
        }
      }
    }
  }
}
```

Figure 5.6: Section of code from the First Experiment's Google App Script

5.3 MainExperiment

In order to assess users in a similar scenario to ones experienced when utilising p≡p in the real-world, a custom application was required. The

front-end is developed in Javascript with a Python-Flask backend that acts as the functionality of the webpage. Several endpoints are exposed like `get_audio` or `get_words` that are requested and displayed on the front-end.

One of the main problems encountered was the handling of each user's session. Initially, during testing, it was noted that changes in one endpoint if timed correctly would overwrite the progress of another, this is commonly known as a "Lost update". This issue was due to the user session being stored in the `session` cookie. When sending off a request the session cookie is included in the header, the code then utilises the data within the cookie to make decisions and alter state, the updated state is then returned as a cookie in the response. If, however, another request is executed before the first has completed, the second request is using the context of the first request and, thus, any alterations made by the first is completely lost.

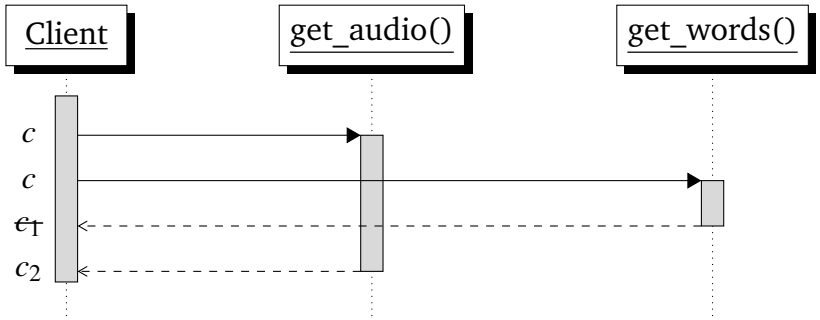

Figure 5.7: Sequence diagram for the 'Lost Update' problem

Figure 5.7 shows this problem visually. As it can be seen the response cookie c_2 is based on the old data of c. When c_2 is eventually returned, it removes the progress of c_1. The restriction of the UI solved this problem. When a button is clicked, all other operations are disabled until a response is received. This technique stopped the possibility of sending off another request while another is still processing. This solution was chosen to keep the complexity of the backend low.

In order to collate data for results later in the process, the state had to be retained. One potential solution would be to format the data and save it as a `.csv` file. However, this is a static way of data persistence and would require alteration every time the data schema was modified. The chosen solution utilised a python object serialisation library known as 'Pickle'[4]. This library saves the instance of a class as a file. This file then can be used to load the instance of the class back into the program. This inclusion allows changes to the object to filter down through the program and keeps the design of the application fluid. For example,

[4]https://docs.python.org/2/library/pickle.html

after initial feedback gathered about the application, it was decided that the inclusion of 'Round times' would be a useful addition. Edits were made to the `experiment` class and the changes propagated through the application; no changes were needed on the save functionality. To distinguish between old and new save formats, a version number was used.

6 Experimentation Results

6.1 Scallion vs GreenOnion

This section compares Scallion and the newly designed GreenOnion ability to search for a large number of potential keys.

Figure 6.2 shows the speed decline as the number of concurrently checked potential keys increases. Alongside the results for the base version of Scallion is the speed for *Scallion Improved*. This is the same Scallion code but with a fix that reduces the severity of the speed decline.

Scallion	
OS	Windows 10 (May 2019)
Version	Scallion 2.1 (Commit: `42b4cb5`[1])
GreenOnion	
OS	Linux Mint 19.1 Kernel: 4.15.0
Hardware	
GPU	Nvidia GTX 750 Ti
CPU	AMD FX-6300

Table 6.1: Testing environment

Figure 6.1 contains a snippet of code with the error. The calls to `parms.ToolConfig.PredictRuntime()` become increasingly expensive as the number of concurrently checked keys increases. Removing this line from the `Console.Write` statement results in a sizeable improvement in speeds as can be seen in Figure 6.2. This was necessary to include as we believe the improved versions show a better comparison between the design of the two tools.

GreenOnion's speed is almost perfectly consistent. It is slower up until around 800 keys. This initial lower speed is due to the overhead of the bloom filter; however, as the number of keys increase, the bloom filter's benefits become apparent. In this test, the bloom filter size was kept consistent at exactly 10,000 elements.

[1] https://github.com/lachesis/scallion/tree/42b4cb555dbf8554b212a2bc3e0ba3d652434ecf

```
1 Console.Write("LoopIteration:{0}  HashCount:{1:0.00}MH
      Speed:{2:0.0}MH/s  Runtime:{3}  Predicted:{4}",
2      loop,
3      hashes / 1000000.0d,
4      hashes/gpu_runtime_sw.ElapsedMilliseconds/1000.0d,
5      gpu_runtime_sw.Elapsed.ToString().Split('.')[0],
6      parms.ToolConfig}.PredictRuntime(hashes * 1000/
           gpu_runtime_sw.ElapsedMilliseconds)
7 );
```

Figure 6.1: Scallion's main print statement

Scallion was unable to handle anything over 2513 concurrent keys. This was due to the search strings being passed via a command-line argument as a regex string. This number of keys hit the character limit of a Powershell command. GreenOnion, however, has been tested to handle more than 1.5 million keys with a slight reduction in performance. This improvement over Scallion is, therefore, substantial.

GreenOnion's code has been made publicly available and can be viewed on GitHub[2].

[2]https://github.com/AidanFray/GreenOnion

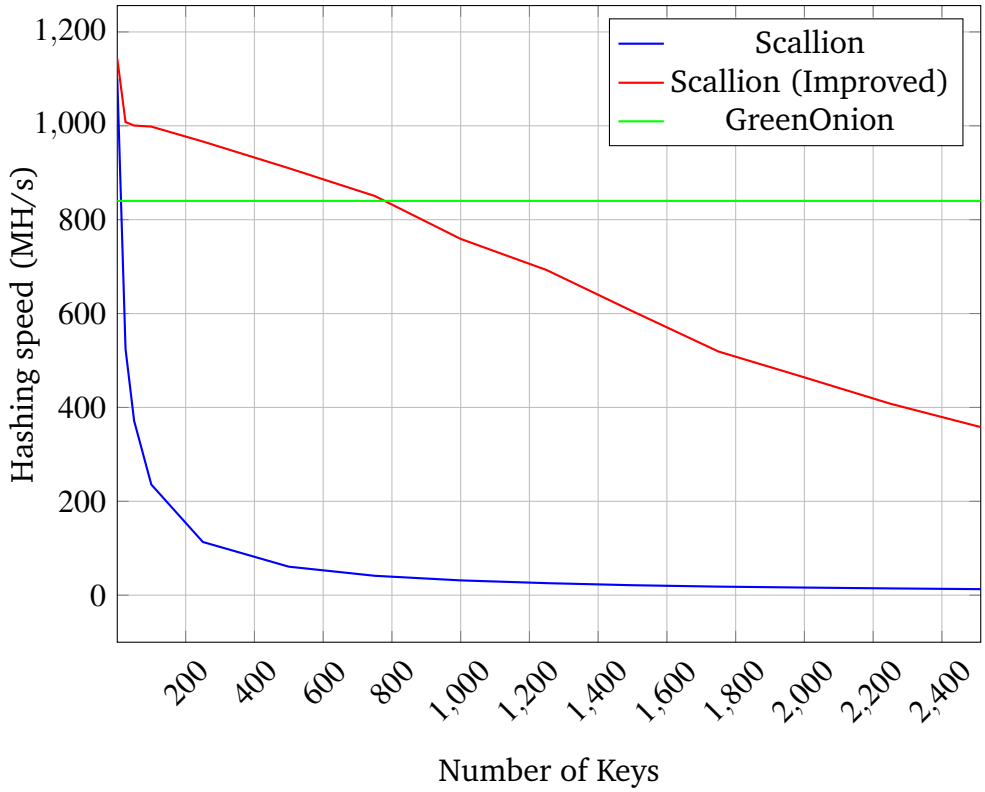

Figure 6.2: Speed comparison between Scallion and GreenOnion

6.2 Experiment 1 - Metric performance

As discussed in Section 4.5.1, the goal of the experiment was to select a set of metrics to be assessed in the following experiment. This section discusses the demographics of participants alongside the subsequent results.

Overall, 104 participants were assessed in this study. Five results were discarded from the set due to either failing the attention questions (See in Section 4.5.1) or having too low of a fluency rating. This dismissal was a necessary process to improve the health of the results.

Table 6.2 contains the demographical breakdown of the reduced set of participants. The average ages of the participants were 37.3 years ($\sigma = 11.67$) with a split of 44.4% of Males to 55.6% of Females. As can be seen, over 60% of participants can be considered highly educated (Bachelor's and up). This level of education is not sufficiently reflective of the general population and therefore, has to be considered when interpreting the results. All participants were sourced from the US; this again requires consideration due to the broad range of dialects present that may bias the results. Further work could investigate the effect of

Gender	Male:	44.4%
	Female:	55.6%
Age:	18-24:	10.1%
	25-29:	20.2%
	30-39:	31.3%
	40-49:	21.2%
	50-59:	12.1%
	60-69:	4.0%
	70-79:	1.0%
Highest Education:	GCSE:	13.1%
	A-Level/O-Level:	19.2%
	Bachelor's degree:	52.5%
	Master's degree:	13.1%
	PhD:	2.0%

Table 6.2: Participant demographics

location and dialect on similar results.

Figure 6.3 shows the average results for the selected matches explained in Section 4.5.1. It can be seen that Levenshtein came out substantially above the rest. The breakdown of the ratings in Figure 6.8 also shows Levenshtein's dominance. Levenshtein has a much more significant proportion of 4 and 5 ratings than the alternatives. This performance may, however, be due to the visual way the comparisons are performed. (Discussed in detail in Section 4.5.1).

Metric	**Average Rating**	σ
Leven	3.66	1.15
NYSIIS	2.92	1.31
Metaphone	2.56	1.32
Phonetic Vec	2.50	1.35
Soundex	2.08	1.12
Random	1.16	0.46

Table 6.3: Average metric performance

Due to the averages of Metaphone and Phonetic Vectors being so close standard deviation was used as the final decider. As can be seen, Megaphone has a slightly lower σ value of that of Phonetic vectors, thus, contributing to the decision to select Metaphone.

Considerations

As discussed in Section 4.5.1, compromises were made to fulfil all requirement with a minimised cost. Levenshtein's abnormal performance further backs up concerns around the visual aspect biasing the performance of the metric. This aspect requires consideration when interpreting

the results.

Even with the discussed issues, the experiment was designed to reduce the number of metrics promptly. This experiment, therefore, has achieved that goal of providing three metrics for the subsequent experiment while balancing between accuracy and expenditure. The results are also an improvement over informal ad-hoc methods. Further work could aim to reproduce this study with the proposed audio-based design.

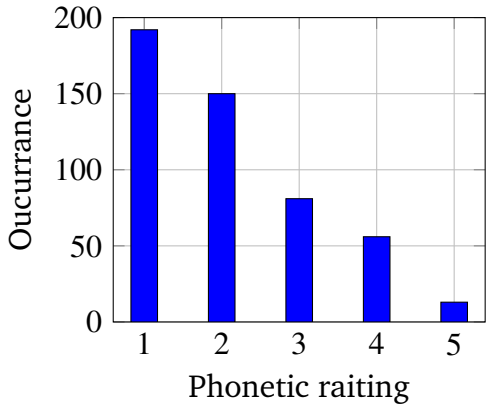

Figure 6.3: Soundex

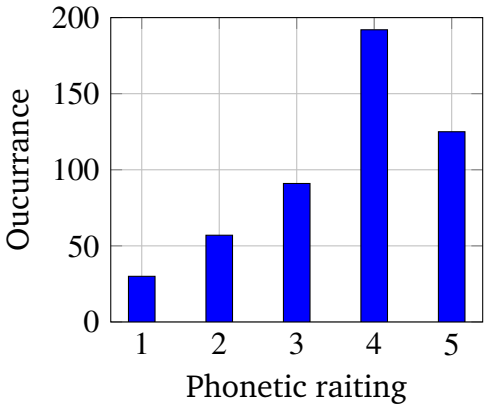

Figure 6.4: Levenshtein

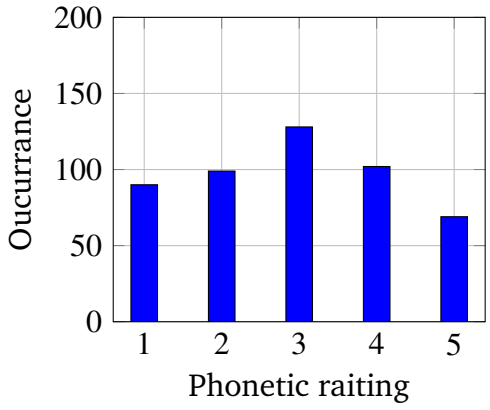

Figure 6.5: NYSIIS

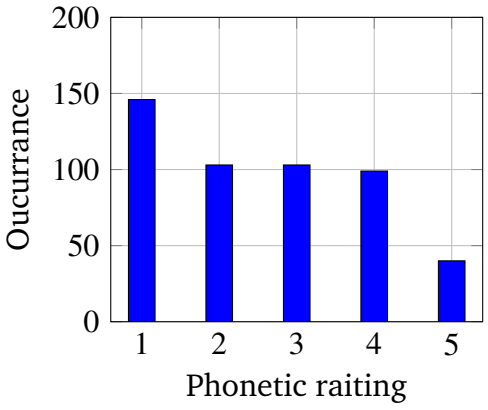

Figure 6.6: Metaphone

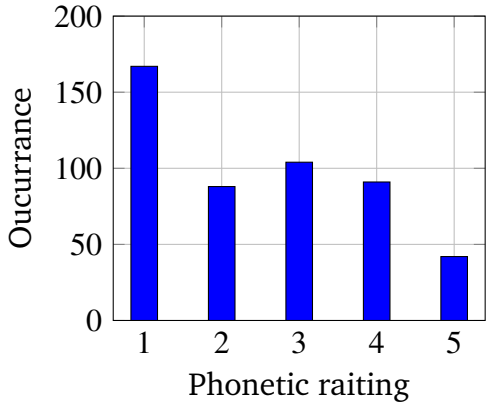

Figure 6.7: Phonetic vector

Figure 6.8: Individual breakdown of results for each metric

6.3 Experiment 2 - Trustword attacks

This experiment aims to quantify the success rate of the proposed attack. Design details for this experiment can be seen in Section 4.5.2. This section presents and discusses the results of the experiment alongside a comparison to relevant literature.

Demographics

Overall, 435 paid participants recruited via Amazon's MTurk were assessed in this experiment. We excluded 66 results; 7 due to being non-native speakers and 59 were discarded for failing the attention metrics (Discussed in Section 4.5.2).

Gender	Male:	50.4%
	Female:	49.6%
Age:	18-24:	12.7%
	25-29:	18.2%
	30-39:	37.4%
	40-49:	17.9%
	50-59:	8.7%
	60-69:	4.3%
	70-79:	0.8%
Highest Education:	GCSE:	13.8%
	A-Level/O-Level:	24.1%
	Bachelor's degree:	51.5%
	Master's degree:	8.4%
	PhD:	2.2%

Table 6.4: Participant demographics

The reduced set of 369 participants had an average age of 36.6 ($\sigma = 11.35$) and consisted of an almost equal split of Male (50.4%) to Female (49.6%). Around 62% of participants had completed a single stage of university (Bachelor and up). This aspect makes this set of participants more educated than the general population. All participants were also sourced from the USA and have rated themselves as fully native English speakers.

As user-agent strings were collected for each participant collations of OS and internet browser versions was made possible. Tables 6.5 and 6.6 contain the browser and OS breakdowns, respectively. Chrome and Windows 10 seem to dominate the proportion of their respective domains alongside the surprisingly large proportion of workers utilising older versions of Windows such as 7 or 8 (15.81%).

Browser	Percentage
Chrome	85.1%
Firefox	12.2%
Safari	1.9%
Other	0.8%

Table 6.5: Internet Browser breakdown

Operating System	Percentage
Windows (All)	74.3%
Windows 7	12.7%
Windows 8	4.1%
Windows 10	57.5%
MacOS X	13.8%
ChromeOS	3.8%
Android	4.3%
iPhone	1.1%
iPad	0.5%
Linux	2.2%

Table 6.6: Operating System breakdown

Results

Table 6.7 contains the break down of results for the experiment. It can be seen that the best metric out of the set was Levenshtein with an overall success of 19.8%. The best performer, when regarding attack strength, as expected, is the **X**○○**X** attack. Levenshtein's **X**○○**X** attack performed the best overall with a success rate of 32.1%. The worst performer was Metaphone, with an average of 16.9% over its three levels of attacks. When comparing the performance of the metrics to the previous experiment, the ordering remains the same with Levenshtein, NYSIIS and Metaphone all ranking in the same order.

Considerations

As previously mentioned, the set of recruited participants are more highly educated than the general population. This combined with the lower average age contributes to an inevitable bias being introduced into the data. An assumption can be made that higher education leads to more skilful interaction with computer-based systems, this, therefore, with the previous assumption in mind means the results of this experiment can be considered a 'best case' scenario. The attacks, therefore, would be expected to be more successful on less technical users. This consideration, however, requires further research and possible empirical

Metric	Successful Attacks	Total Attacks	Success Rate
Levenshtein	218	1101	19.8%
○○○○	34	358	9.5%
X○○○	59	353	16.7%
X○○**X**	125	390	32.1%
Metaphone	181	1072	16.9%
○○○○	38	345	11.0%
X○○○	57	375	15.2%
X○○**X**	86	352	24.4%
NYSIIS	209	1114	18.8%
○○○○	36	385	9.3%
X○○○	72	375	19.2%
X○○**X**	101	354	28.5%
Overall	608	3287	18.5%

Table 6.7: Success rates for simulated attacks

data to determine the validity of the assumption.

During this study, the main task was to perform the authentication ceremony a set number of times. In the real-world, the ceremony is a secondary task due to it not strictly being required. This aspect, alongside the potential for users to downplay the possibility of attack, leads to the conclusion that the success rates could be much higher in a real-world setting. The assumption is that these results display the success rate for attentive users where they make sure to double-check with their authentication partner multiple times. It is, therefore, assumed that the majority of users aim to complete the task as quickly as possible, if at all.

Comparison to alternative literature

This section compares the results of the experiment to similar literature.

Work by **R. Kainda, I. Flechais** and **A. Roscoe**[17] (previously discussed in Chapter 2) is the most comparable to this experiment. They were the only study to use exactly 4 words but differed on the size of the dictionary (1024 words) and how a near-match were calculated. Near-matches were a difference in a single word, but without the consideration for similarity, this is the unique aspect considered with this experiment.

However, with those aspects in mind attacks on words encoding received an overall success rating of 3.3%, considerably lower than that

of this study. In the worst-case, our simulated attacks achieved almost 3 times as many successes compared to this study.

Other relevant work by **S. Dechand *el al.*** [15] also discussed in Chapter 2 assess the success rate of attacks on words. Their experiment is less similar as 14 words per attack were assessed with each participant, where the comparison was performed visually. However, the success was 8.78%. As 10/14 words were kept static, this result is more comparable to the highest attack strength (**X**○○**X**). This, therefore, leads to another 3 fold improvement in the success rates of our simulated attacks.

The final relevant paper to compare is that of **J. Tan *et al.*** [16]. This paper had the highest number of words assessed with 16 overall. With it assessing the same wordlist and attack strength as the work by **S. Dechand *et al.*** [15], the attack success was 14% overall. This success rate compared to the highest attack assessed in the experiment results in another sizeable difference.

In conclusion, all related literature had much lower attack success rates that the results presented in this experiment. With all papers using similarly designed wordlists, it highly suggests that the deficiencies in Trustwords are the cause for the substantial increase in attack success. Alongside this, the consideration for similarity when calculating near-collisions could have also resulted in the higher attack success rates. This aspect is also a unique consideration compared to the available literature.

6.4 Average number of near-collision keys for each metric

In order to predict the effectiveness of the attack over a wide variety of keys, the number of average near-collision keys requires computation. Table 6.8 shows the average number of potential collision keys for each metric. The “Compute Time” is the computation time required for a single mid-range (2000MH/s) GPU.

As can be seen, metrics like Levenshtein have a low overall average and would require much more attacker resources to become feasible. On the other hand, Metaphone has attacks that could be computed on average in under 2 days. The success rate for this attack type in the second experiment was 11.01% on average. This means on commodity hardware commonly found in the home; an attacker could compute a key in less than two days that could succeed more than 10% of the time. This aspect is substantial and highlights weaknesses in the Trustword system.

Metric		Near-collision keys	Compute Time
NYSIIS			
	○○○○	1963.07	27.19 days
	x○○○	448.37	119.14 days
	x○○**x**	98.07	1.49 years
Metaphone			
	○○○○	27138.78	1.97 days
	x○○○	2822.74	18.91 days
	x○○**x**	339.84	157.45 days
Levenshtein			
	○○○○	293.92	182.17 days
	x○○○	102.75	1.43 years
	x○○**x**	35.53	4.18 years

Table 6.8: Each metric's average number of near-collision keys and average compute times

6.5 Distribution of vulnerable keys

As discussed in the design of the second experiment (See Section 6.3), the attacks sampled from a list of 'vulnerable keys'. These vulnerable keys were created using real keys extracted from PGP key servers. These keys were then randomly sampled and used to create the combined key, as discussed in Section 2.2.2. 100,000 random key pairs were created and used as the sample[39]. Any vulnerable keys in this set were recorded and used in the experiment.

Attack Type	Metric	Vulnerable %
x○○**x** (152)		
	Levenshtein	4.295%
	Metaphone	24.095%
	NYSIIS	10.441%
x○○○ (1525)		
	Levenshtein	0.827%
	Metaphone	15.569%
	NYSIIS	4.341%
○○○○ (15250)		
	Levenshtein	0.157%
	Metaphone	10.202%
	NYSIIS	1.824%

Table 6.9: Number of vulnerable keys per metric

Table 6.9 contains the breakdown of the number of vulnerable keys.

The takeaways from this table are the extremely low values for Levenshtein in the higher attacks. This phenomenon is due to the low number of overall matches produced by Levenshtein, meaning the metric is not effective in the vast majority of cases. However, the standout metric is Metaphone with a sizeable proportion of vulnerable keys; it was the worst performer when assessed against real users but only by as little as 2%. If you take into consideration, the number of vulnerable keys detected in the set alongside the average amount of near-collision matches in Table 6.8 Metaphone appears to be the most usable metric overall.

6.6 Generation of keys

In order to demonstrate that the generation of the near-collision keys is not only theoretical, the actual generation was performed using GreenOnion. A random metric was selected from the three carried forward from the first experiment. Alongside this, a random uncontrolled key was generated as the simulated attack target (See Appendix A for the armor public key). Then for each attack level (OOOO, **X**OOO, **X**OO**X**), a key was computed. NYSIIS was chosen as the selected metric. To expedite the process, an initial search was performed for controlled keys that had the highest number of near-collisions; this was due to the demonstrative purpose of this experiment. Producing near-matching keys from key pairs with a very small number of potential matches adds very little to this demonstration. This, however, results in a best-case scenario for key computation times.

Tables 6.10, 6.11 and 6.12 contain the parameters and results on the computation. Due to the varying availability of resources, each computation has a varying hashing speed. This variation has been normalised by the introduction of the "GPU Day" metric. A GPU day is simply the computation time required to reach the same conclusion on a single 2000 MH/s GPU.

As can be seen most notably from Table 6.12, the generation of the actual Trustwords differs in only 3 characters making it arguably very similar. When linking this generated key to results of the second experiment (See Section 6.3) computation of 2.93 days on a single mid-range GPU can achieve around a 28% success rate when assessed against actual users. This time, however, has been improved due to the random search for a controlled key with a more favourable number of near-collision keys. Even with the least restricted attack type presented in Table 6.10, the computation of half a day can provide almost a 10% success rate when attacking users.

Potential near-collision keys	142,296
Hashing speed	4000 MH/s
Estimated computation time	9 hours
Actual computation	6.41 hours
GPU Days	0.52
Original Trustwords	BELL GRIND ALGERIA ANNULI
Actual Trustwords	BOIL GRAND ALGER ANNUL

Table 6.10: NYSIIS - OOOO

Potential near-collision keys	133,200
Hashing speed	8000 MH/s
Estimated computation time	4.2 hours
Actual computation	5.12 hours
GPU Days	0.853
Original Trustwords	ASSUMING SONOMA DENS KEENER
Actual Trustwords	ASSUMING SUMMON DENNI CONNOR

Table 6.11: NYSIIS - XOOO

Potential near-collision keys	16,464
Hashing speed	6000 MH/s
Estimated computation time	2.16 days
Actual computation	23.44 hours
GPU Days	2.93
Original Trustwords	VOCATION BORE TANN ANTE
Actual Trustwords	VOCATION BARE TONE ANTE

Table 6.12: NYSIIS - XOOX

7 Conclusions

This chapter collates the findings of the paper, evaluates the paper's success against the research questions and discusses potential further work.

7.1 Research Question 1

Research Question 1 asks: *"What are the best performing schemes to quantify phonetic similarity?"*. This paper has discussed several metrics used to quantify phonetic similarity (See Section 4.2). Most metrics were chosen due to their high utilisation and occurrence in literature. The only exception to this is the presence of Phonetic Vectors. Performance of these metrics were then quantified in a direct study that selected the top three from a set of five (See Section 4.5.1). These newly selected metrics were then assessed in an attack context with their performance being indirectly quantified (See Section 4.5.2).

The top three metrics (in descending order) were: Levenshtein, NYSIIS and Metaphone. We found that Levenshtein was the most aligned with how people rate phonetical similarity. Levenshtein had a considerable proportion of 4 and 5 ratings (64%) compared to the alternatives. Levenshtein was also the best performer in the attack context with it having the highest success rate. This corroborates work showing that words with similar spelling tend to share phonetic similarity [36]. Alongside this, we found Soundex was the least aligned with the human model of similarity with an average rating of just 2 out of 5. This also corroborates findings that highlighted significant issues with the accuracy of Soundex's phonetic representation (See Section 2.1.3).

We believe the outcome of the first experiment were inflated by the way the results were presented (as discussed in Section 4.5.1). However, as the performance of the metrics in the second experiment emulated the first with Levenshtein being the best performer alongside Metaphone being the worst, shows that the results of the first study maintain a desirable level of accuracy. Improvements can be made with the proposed design discussed in Section 4.5.1, but, we believe the first experiment has achieved its goal of providing a low-cost way to reduce the number

of selected metrics for the subsequent study.

7.2 Research Question 2

Research Question 2: *"What is the time and computation complexity required to generate a 'similar' keys for a targeted key pair?"* has been discussed by the project alongside the actual generation of keys as shown in Section 6.6. Furthermore, the distribution of key permutations on a set of real-world keys were discussed in Section 6.4 and 6.5. To achieve this, the tool GreenOnion was developed. Inspiration for GreenOnion was gathered from a currently available tool known as Scallion. Experimentation results were gathered and displayed in Section 6.1 that showed the significant increase in the number of near-collision keys GreenOnion can concurrently search for. GreenOnion is able to handle in excess of 1.5 million keys. This capability is in comparison to the maximum of 2513 keys the alternative; Scallion can compare. We believe this is due to the implementation of the bloom filter. Its unique characteristics of $O(k)$ complexity, regardless of the number of elements in the structure allows a huge number of concurrent checks to occur.

7.3 Research Question 3

Research Question 3 *"What percentage of attacks successfully deceive a user?"* was shown by the results presented in Section 6.3. The weakest attack strength and ones that can be computed by a single mid-range GPU for a single week achieved success rate ranging from 9.35% - 11.01%. The middle attack involving the use of 70 mid-range GPUs for a week had success rates of 15.20% - 19.20%. Finally, the highest attack strength that theoretically involved the use of 700 mid-range GPUs computing for a week has success rates: 24.43% - 32.05%. Therefore, showing the number of attacks required to deceive a set of users.

7.4 Research Question 4

Research Question 4 states *" Is the recommended minimum number of four Trustwords enough to provide a basic level of security?"*. The paper has investigated this through the creation of an attack and quantification of its success on actual participants. Findings have shown that an attack computed in half a day can have a success rate of about 9.35%. A user, therefore, could initiate this attack on-mass with hardware present

inside the average gaming computer. On the other hand, the best level of attack success was shown to be 32.05%.

Furthermore, comparison of these results to similar literature shows a considerable increase in success rates for the attack present in this paper. We believe this is due to the consideration of 'similarity' when forming attacks. This was an aspect missing from the literature. We also believe the effect of considering similarity is further enhanced due to the weaknesses present in Trustwords such as homophones, dual mappings and the wordlist size. Therefore, we believe that four Trustwords as a minimum has not provided a basic level of security towards users.

7.5 Further work

Trustword improvements

The first area of proposed work could be recommendations into how to improve Trustwords backed by empirical evidence. We feel the most promising avenue would be the utilisation of Phonetic Vectors unique qualities to identify words of low quality by measuring very low vector distances. For example, present in the dictionary are the words `THERE` and `THEIR`, these could be massively improved upon and could result in a better quality word list. Moreover, utilising the quantification of dissimilarity could be used to create a dictionary of maximised phonetic distance. This phonetically distinct wordlist could then be assessed in a similar way with its success rate quantified against users.

Similar metrics performance

Another area of further work is the comprehensive assessment of algorithms used to assess phonetic similarity. The algorithms assessed in this work are a very small proportion of potential options. Thus, further work could assess the performance of metrics against each other to determine the one that works best with human models. A conclusion could be reached by running the improved experiment discussed in Section 4.5.1, where it could be repeated with an increased number of metrics. Other elements for consideration could be age, location and dialect as variables that affect a metrics performance.

GreenOnion optimisations

A minimal amount of work was allocated to improving the performance of GreenOnion as low-level optimisations are very time-consuming. A project, therefore, could aim to improve on the performance already recorded in this paper. This project could improve the performance of the attack and allow for more effective keys to be computed in less time.

7.6 Final Remarks

Overall, the project has demonstrated a potential attack on p≡p 's implementation of Trustwords. An attack was proposed that utilised phonetic similarity algorithms to exploit the weaknesses in the Trustword dictionary to generate near-collision keys. To achieve this, a tool was created to compute a massive number of keys concurrently. It was based on a known tool but improved substantially on its key searching performance. The performance of similarity metrics was compared alongside an experiment assessing participants fallibility to the proposed attack. Results showed as much as a 32.05% success rate for the best attack. The main project aim was to show that the recommend minimum number of four Trustwords was insufficient to provide a basic level of security. We believe we have demonstrated that four Trustwords is too low to provide enough security for general use-case and the minimum provided words should, therefore, be increased.

A Randomly generated uncontrolled key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBF0vHaYBCACxOy7RacHIprgaUo66vGmIB2VIaSwToliuaEBxOAloD+wTB/T9
TOKgAMLWMgK2z3/tOZj2l5k9ATUr0nZ9iBgew5Ih3ykOp+OkE9dh4NTD3EJz5UWo
M1r6scPhby92zMBqKi0iPFlPcvk+eg+SPusxIp+Vn16ALoB2pwGQvG+qoFEJfdVP
nwSAWJo1mcd+TCxgIMqe11FXDzxpd4BEpMXsHOB8NH/TUiI57sxqT8A2HnnyWo2i
5vs93VAP8rTTIRfGelW2c3oqhV9XhXbvVpJPmvkTxM/SshV1L5HhYr1TUAkAQw6w
RUZqooysgZEDqQw58lFQ7C9plCHOpXIEhbE7ABEBAAG0HEpvaG4gSm9obnNvbiA8
am9obkB3b3JrLmNvbT6JAVQEEwEIAD4WIQRQZ0ukmi5QUEz7KB3FwHKSVKqpwgUC
XS8dpgIbAwUJA8JnAAULCQgHAgYVCgkICwIEFgIDAQIeAQIXgAAKCRDFwHKSVKqp
wg4YB/9knveazSM4yN3EJY7HRZe5PVA5r0GGDlEDMte+surPKLsV/T39SNl8qTH+
DXnOyC4rt1kykTJP+ULcKqGkVRnH2XYZP/W6E9wwBPjoKHaWyn8MP0ix5kgkq6Ld
ZWEQ2pUwCsAhWv8K/ybjQ+Li484JKbrz/GRALp8qU3hZgCo/Fw3sD2KcTPU1v8qU
av9G2N5su10W7G0F1ozszFit+r2iinoZtGSddBPG+pIUEtZoBn4V2FRRSO1UY/TU
jGf2BLsk+ZkNJdx5x6IEU9CQ+ivPuKtupVu9ySKqE2Qfmn5lh9iAjYDN3XxUQfCf
braGoC9JLnL5QmT1JaQXfxkIHilyuQENBF0vHaYBCADDp/u8D/TG5d5HmabmgbBF
3R2d3J00vt7VQQ5sIRMLyP0cx6qy2bTIy9mgfU3/WcrhRp728gBo+NnUfxyiTb5R
u7AbntlkerF3dh0JLm5eIossPW37QXn3Ce4Tv4ixGRLbsGKTu6h0lP41FM5VcqV9
9NE1iml3WyngnsFdKeUYqPW4QY+ydyjpSyblbQTMptIKG2llCmsd9k+hb8nnGx/d
GvyGKaZ270iVGY2im/9VP2VwzqXlFxxx6Dux4egJJMMAy5xkzBKf+Szmm1pMJkt2
4sBrwmLnuUObgsAkq8K+MqEpWeFi6aWsh7RnY8T0SEb/3QUwG12V77ePhrarBwmf
ABEBAAGJATwEGAEIACYWIQRQZ0ukmi5QUEz7KB3FwHKSVKqpwgUCXS8dpgIbDAUJ
A8JnAAAKCRDFwHKSVKqpwoW3CACeKIcQFuVv2vpUHq/4ATH0atPYqR1NbJTegE+v
XZyrNHtmRF4V+kdx/1KX9jsIY2TjyxbhTsrtMFkMZMfAVjXVoS4y8OkwX5IJWgHC
0JQIHQrvaIGfI1yuVrNvWYyPXFxNtbntrgTd8JKYGKYaTqXNZ2r1ozbZElp7fQtR
rWNUGJYYo+aHHLM+Aot5k+Z3YqHifuPSnCEYt/GxnMhYq1IL3LSZ96hsgwqeFoLP
+v4RsvY52Yb5tiOzxZwikALOJdM2aAumxq8R1nYHiswWxotw2WkKFlB7gfLpbMWs
Y+o4J7QKSqpVTrUYBTPF1F4kBWOMY10kpeCDbZ1YwJOZJf8P
=aHBu
-----END PGP PUBLIC KEY BLOCK-----
```

B Computed Attack Keys

B.1 NYSIIS - 0 Static

B.1.1 Controlled Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mI0EXS8lIAEEAMWEJ+NGB/ukiUijzdT+DvzVwouWV/j4Rw78X9YdgJP0ExT4ncHG
TJbOKjEMEI9bb/oYMLF/434ufScYNci+rL/3RVY8CcjEv5l0UGX+udZ78c7Z/vf1
zFr2MQAglr9M8599JwUY25LiNKzDYc87B16hqRV8hRaXHfAFFHBxRsgBABEBAAG0
HUF1dG9nZW5lcmF0ZWQgS2V5IDx1c2VyQENWRUQ+iNQEEwEKAD4WIQSlBU278MbT
iVM5h6GNohojnp5YQgUCXS8lIAIbLwUJAeJfkAULCQgHAgYVCgkICwIEFgIDAQIe
AQIXgAAKCRCNohojnp5YQvxSA/0bjL+4me3hLjrM+8IckKABCRdRYYRG7svbln9q
ULUngsXMpbIMoHFg1ePpT2lniiHtQhDC88ZW1J5yeVZN1cfJhRsByyKlkzG3Dcy7
5wKonbVz1LSYpIDq/DNNP0eINzuVNc1rxqRXPGAh6NHnmZN5MqxLNyu0DSX9oJHm
KbbtLQ==
=FQCu
-----END PGP PUBLIC KEY BLOCK-----
```

B.1.2 Computed key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQEOBF0zhx4BCAD0iC55GShgArPzKZdn6we2mrkxS8o8fI7HdWXnLoRAta1Q1j9z
U+mYU9tGYb0yjqIARjjUK2qnq0CsGZJpB65AhbrOsDLv6dmI68tuiT4xjmzcM6+6
6M/EY5E5hYpZeurDJ8rSYrfu/v9P0bV78VnM17uUl4PTyEBmbJsE9kszq4MXd5yz
G+ibBPBvnBWSlRQzyCFMgTL8gQH36XJUj7jgxTKoJWzli/QtNcMOjwSQtEZxFWVK
+qsJjYUClpg9Kkow1LqJmEc1Tj98EipypDtdwNQhn4wHQVgu7SliaNNmrR8MpP9o
gL9hhfafx5FsE7v4zHhzu+ubR+c7KnUlwJyfABkBhI5atQAeSm9obiBEb2UgPGNo
YW5nZV9tZUBlbWFpbC5jb20+
=
-----END PGP PUBLIC KEY BLOCK-----
```

B.2 NYSIIS - 1 Static

B.2.1 Controlled Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mI0EXS8o2AEEAKQmxot2Z5XFrNjg4zXBfvLsTzoO3vZeEI9l8+U2X8aq5KVQJpPU
BRSfqQuMZvA1zh3vH3w4I/vR4nXCNw/hPCi0B/DckUe6IJPZJfMiQfT1Xt16ZEms
W53CBiNqnt5xruo4wKwoKGuHELtPcD4K/iBIFt6n3UTi+R8xU4WMc+PjABEBAAG0
HUF1dG9nZW5lcmF0ZWQgS2V5IDx1c2VyQENWRUQ+iNQEEwEKAD4WIQTFXc8VXOqe
13ib81dcbeH3Z8A13wUCXS8o2AIbLwUJAeJb2AULCQgHAgYVCgkICwIEFgIDAQIe
AQIXgAAKCRBcbeH3Z8A13wItA/kBbfc6kxtLumDQiGpcIzgWmM9UhCn0V8ny9iB2
gt4V8bImpC3qicj6+KywsEsvFIgsJO0UcNeoREKanpXxPf6T7tvBUTLQjPJFedsn
J57996JiZ9COQ4lEg2AvfAKy1alKrMs0Xy4hWrHY9cJWs1a7KpbrsioHd4vDRgMS
2yRn/w==
=krrv
-----END PGP PUBLIC KEY BLOCK-----
```

B.2.2 Computed key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQEOBF0yA4QBCAC78Lk2k9BdUEufpqll4D60iAuYVrxVs3lpjwP2hpztBFFw8vY4
26uW1C7MPo9l4Oteks5626hwAAgI1/YHFwWpBDdTWYEnVGrjv5v0Juz5IZc7kiwp
Ui2XWJ7I3VpE5nljFyzaQMGaS5Cvmtco5cjin1Tx486jieGBt0loQQdtiJ3gEAFT
XlUvwIQSdYWY2F8xWKg5aJoDFEZN74KfwQqB7hx/53VHyzOJGygnvvpMcJaX3c8E
V3t7n90WvXJWND7bn6HIJrnFKyRJsKHneL2n0wGd/N0NLCuLTvNK+uQ7sXGNZox1
Qa4a1guiSdaq9NO5mX+tOYox3/onzpshJgVDABkBvckntQAeSm9obiBEb2UgPGNo
YW5nZV9tZUBlbWFpbC5jb20+
=
-----END PGP PUBLIC KEY BLOCK-----
```

B.3 NYSIIS - 2 Static

B.3.1 Controlled Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mI0EXS8s+gEEANkph53UuQkNVFhYimrzH0bXBp/Y3InV9p5MP09TZxtTOf44JBPp
MCFMvgFJpVK3D5PQ2h7fW1lNb41THkiGC7Jk+F7gd+beqH5kaaKmC+RNAF0aZ5av
2al5JvtHT0Z8/1r8bxfp33WLmQXK0/9EwL4HvERRGh7wZ5WemvCrvZPXABEBAAG0
HUF1dG9nZW5lcmF0ZWQgS2V5IDx1c2VyQENWRUQ+iNQEEwEKAD4WIQSjUV9vxlLP
LPE78Z2TulybLqhYHAUCXS8s+gIbLwUJAeJXtgULCQgHAgYVCgkICwIEFgIDAQIe
AQIXgAAKCRCTulybLqhYHONrBAC8uh/K2Adq/o/Qe/EVM5JpT6ETE1WlAj2iUMWB
MAmdjgiQlRyBCTd4Lt9xFsbnvQG9pj4GVHosXjVpuA/1NxyS+DE4EkBDzXA3NuR8
0djhyD27VSsn2q1j8bB1wxO7sLbjo30SIBG2wUaft5DVJiBGG69kh4pt+aKKOcjb
LLPldw==
=dCl/
-----END PGP PUBLIC KEY BLOCK-----
```

B.3.2 Computed key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQEOBF0wxGEBCACsqQN92G3IizweQHw42OhUM5ZySKTvzPqVhAeCb2ETSBDZFbGN
lHYPa6dytXwLbU6kdlt8kgxbk75iPJsCSPLAUzKvlVd2rkTu5xtKPFqDTRvbjOjf
Ik8zJimXJGZNYVrMUqkfPR2FtVC4VUJXtML8voj/1lcLxTWRcYjuNQuSRNhJQtUP
ViZMTj97IO894lVg+ciZXEZkzHzlriqeLqwTupQxvzqjmObmeH1NCPSfGcTSNIBu
cNG/DrXTXsVpZPUilIRsXTtc2FM9iY49Zjv01KdzsJlV97JVyghYFFCEe1xq7Twp
U9TOzypXD4NdTKivFZUNT4jdG2HoJE4IcZ3NABkBh0qYtB5Kb2huIERvZSA8Y2hh
bmdlX21lQGVtYWlsLmNvbT4=
=0nU6
-----END PGP PUBLIC KEY BLOCK-----
```

C Trustword Attack

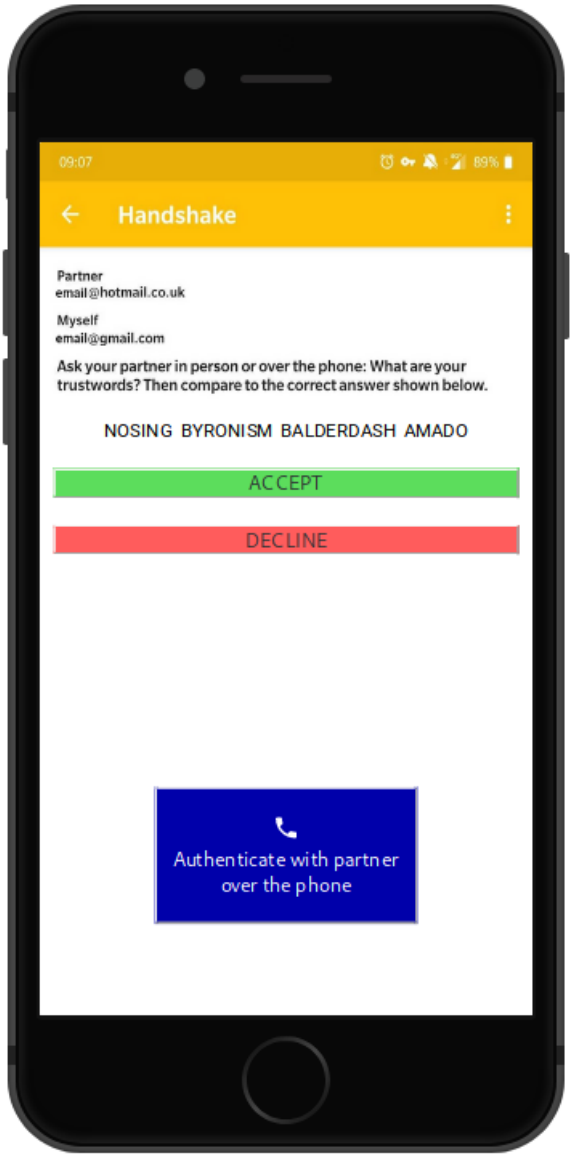

Figure C.1: Experiment UI

Bibliography

[1] G. A. Miller, 'The magical number seven, plus or minus two: Some limits on our capacity for processing information.', *Psychological review*, vol. 63, no. 2, p. 81, 1956.

[2] J. Callas, L. Donnerhacke, H. Finney and R. Thayer, 'Openpgp message format', RFC 2440, November, Tech. Rep., 1998.

[3] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau and A. Narayanan, 'An empirical study of namecoin and lessons for decentralized namespace design.', in *WEIS*, Citeseer, 2015.

[4] 'Whatsapp encryption overview - technical white paper', Dec. 2017.

[5] A. Perrig and D. Song, 'Hash visualization: A new technique to improve real-world security', in *International Workshop on Cryptographic Techniques and E-Commerce*, 1999, pp. 131–138.

[6] C. Ellison and S. Dohrmann, 'Public-key support for group collaboration', *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 4, pp. 547–565, 2003.

[7] Y.-H. Lin, A. Studer, Y.-H. Chen, H.-C. Hsiao, L.-H. Kuo, J. M. McCune, K.-H. Wang, M. Krohn, A. Perrig, B.-Y. Yang *et al.*, 'Spate: Small-group pki-less authenticated trust establishment', *IEEE Transactions on Mobile Computing*, vol. 9, no. 12, pp. 1666–1681, 2010.

[8] H.-C. Hsiao, Y.-H. Lin, A. Studer, C. Studer, K.-H. Wang, H. Kikuchi, A. Perrig, H.-M. Sun and B.-Y. Yang, 'A study of user-friendly hash comparison schemes', in *2009 Annual Computer Security Applications Conference*, IEEE, 2009, pp. 105–114.

[9] F. Patman and L. Shaefer, 'Is soundex good enough for you? on the hidden risks of soundex-based name searching', *Language Analysis Systems, Inc., Herndon*, 2001.

[10] L. Philips, 'Hanging on the metaphone', *Computer Language*, vol. 7, no. 12, pp. 39–43, 1990.

[11] A. Parrish, 'Poetic sound similarity vectors using phonetic features', in *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[12] N. Chomsky and M. Halle, 'The sound pattern of english.', 1968.

[13] P. Ladefoged, 'The measurement of phonetic similarity', in *INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS COLING 1969: Preprint No. 57*, 1969.

[14] A. Bradlow, C. Clopper, R. Smiljanic and M. A. Walter, 'A perceptual phonetic similarity space for languages: Evidence from five native language listener groups', *Speech Communication*, vol. 52, no. 11-12, pp. 930–942, 2010.

[15] S. Dechand, D. Schürmann, K. Busse, Y. Acar, S. Fahl and M. Smith, 'An empirical study of textual key-fingerprint representations', in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 193–208.

[16] J. Tan, L. Bauer, J. Bonneau, L. F. Cranor, J. Thomas and B. Ur, 'Can unicorns help users compare crypto key fingerprints?', in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ACM, 2017, pp. 3787–3798.

[17] R. Kainda, I. Flechais and A. 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*, ACM, 2009, p. 11.

[18] M. Shirvanian, N. Saxena and J. J. George, 'On the pitfalls of end-to-end encrypted communications: A study of remote key-fingerprint verification', in *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACM, 2017, pp. 499–511.

[19] E. Uzun, K. Karvonen and N. Asokan, 'Usability analysis of secure pairing methods', in *International Conference on Financial Cryptography and Data Security*, Springer, 2007, pp. 307–324.

[20] M. M. Olembo, T. Kilian, S. Stockhardt, A. Hülsing and M. Volkamer, 'Developing and testing a visual hash scheme.', in *EISMC*, 2013, pp. 91–100.

[21] D. Loss, T. Limmer and A. von Gernler, *The drunken bishop: An analysis of the openssh fingerprint visualization algorithm*, 2009.

[22] A. Karole and N. Saxena, 'Improving the robustness of wireless device pairing using hyphen-delimited numeric comparison', in *2009 International Conference on Network-Based Information Systems*, IEEE, 2009, pp. 273–278.

[23] P. Juola, 'Whole-word phonetic distances and the pgpfone alphabet', in *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96*, IEEE, vol. 1, 1996, pp. 98–101.

[24] M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik and E. Uzun, 'Loud and clear: Human-verifiable authentication based on audio', in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, IEEE, 2006, pp. 10–10.

[25] N. Haller, 'The s/key one-time password system', 1995.

[26] K. Rieck, 'Fuzzy fingerprints attacking vulnerabilities in the human brain', *Online publication at http://freeworld. thc. org/papers/ffp. pdf*, 2002.

[27] M. Shirvanian and N. Saxena, 'Wiretapping via mimicry: Short voice imitation man-in-the-middle attacks on crypto phones', in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2014, pp. 868–879.

[28] D. Mukhopadhyay, M. Shirvanian and N. Saxena, 'All your voices are belong to us: Stealing voices to fool humans and machines', in *European Symposium on Research in Computer Security*, Springer, 2015, pp. 599–621.

[29] S. Chen, K. Ren, S. Piao, C. Wang, Q. Wang, J. Weng, L. Su and A. Mohaisen, 'You can hear but you cannot steal: Defending against voice impersonation attacks on smartphones', in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2017, pp. 183–195.

[30] Z. Wu, N. Evans, T. Kinnunen, J. Yamagishi, F. Alegre and H. Li, 'Spoofing and countermeasures for speaker verification: A survey', *speech communication*, vol. 66, pp. 130–153, 2015.

[31] R. L. Peterson and B. F. Pennington, 'Developmental dyslexia', *The Lancet*, vol. 379, no. 9830, pp. 1997–2007, 2012.

[32] *Mysql 5.5 reference manual :: 12.5 string functions and operators*. [Online]. Available: https://dev.mysql.com/doc/refman/5.5/en/string-functions.html#function_soundex.

[33] *Oracle - database sql reference*, Jul. 2005. [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions148.htm.

[34] *F.15. fuzzystrmatch*. [Online]. Available: https://www.postgresql.org/docs/9.1/fuzzystrmatch.html.

[35] *Metaphone*. [Online]. Available: https://www.php.net/manual/en/function.metaphone.php.

[36] G. P. Hettiarachchi and D. Attygalle, 'Sparcl: An improved approach for matching sinhalese words and names in record clustering and linkage', in *2012 IEEE Global Humanitarian Technology Conference*, IEEE, 2012, pp. 423–428.

[37] D. Difallah, E. Filatova and P. Ipeirotis, 'Demographics and dynamics of mechanical turk workers', in *Proceedings of the eleventh acm international conference on web search and data mining*, ACM, 2018, pp. 135–143.

[38] M. Cherubini, A. Meylan, B. Chapuis, M. Humbert, I. Bilogrevic and K. Huguenin, 'Towards usable checksums: Automating the integrity verification of web downloads for the masses', in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2018, pp. 1256–1271.

[39] D. Norman, *Pgp-key-utilities*, Aug. 2019. [Online]. Available: https://github.com/DavidJNorman/pgp-key-utilities.