Ronja Lindholm

Honey Encryption: implementation challenges and solutions

Master’s Thesis in Information Technology

June 1, 2019

University of Jyväskylä

Faculty of Information Technology

Author: Ronja Lindholm

Contact information: `ronja.lindholm@gmail.com`

Supervisor: Andrei Costin

Title: Honey Encryption: implementation challenges and solutions

Työn nimi: Hunajasalaus: toteutuksen haasteet ja mahdollisuudet

Project: Master's Thesis

Study line: Faculty of Information Technology

Page count: 61+0

Abstract: Most encryption techniques have one essential problem: they are vulnerable to brute-force attacks. In 2014, a new defence mechanism was introduced against brute-force attack and this mechanism was called Honey Encryption. Honey Encryption has been cited in several articles and it has demonstrated potential to be an additional security method alongside conventional encryption. This thesis will offer an overview of the current situation of Honey Encryption and it starts by explaining and discussing the challenges that have been brought up. These challenges include, among other things, the difficulty of producing the honey messages and identifying user typos from an attack. Some of the solution ideas that have been introduced since the introduction of Honey Encryption are discussed as well. There have been some new types of data that could be used with Honey Encryption, such as genomes and multi-dimensional data such as images and videos. These data types are discussed in connection with the use of Honey Encryption. As a last measure to provide overall understanding of the current situation of HE, this thesis provides an implementation model for Honey Encryption in a theoretical level, with some practical programming parts as well. The model is offered by going through the Honey Encryption scheme step by step. Together, these three steps: challenges, new data types, and an implementation model are meant to give a broad view of the situation of Honey Encryption gathered in one place.

Keywords: Information security, Encryption, Honey Encryption, Cryptography, Brute-force, Decoy, Sweetword, Password-based encryption, Symmetric cryptography

Suomenkielinen tiivistelmä: Useimmilla salaustekniikoilla on yksi olennainen ongelma: ne ovat haavoittuvaisia väsytyshyökkäyksille. Vuonna 2014 esiteltiin uusi puolustuskeino väsytyshyökkäystä vastaan nimeltä hunajasalaus. Hunajasalausta on siteerattu useissa tietolähteissä sittemmin ja se on osoittanut potentiaalia ollakseen lisäsuojaus perinteisen salauksen rinnalla. Tässä gradussa esitellään yleiskatsaus hunajasalauksen nykytilanteesta ja se aloitetaan käymällä läpi esiintulleita haasteita. Näitä haasteita on muun muuassa hunajasanojen luonnin vaikeudet ja käyttäjän kirjoittamien virheiden tunnistaminen hyökkäyksestä. Joitakin ratkaisukeinoja, joita on esitelty alkuperäisen artikkelin julkaisun jälkeen, käydään lävitse. Jonkin verran on tutkittu uusia tietotyyppejä, joita voitaisiin hyödyntää hunajasalauksen kanssa, esimerkiksi genomit ja moniulotteinen data, kuten kuvat ja videot. Nämä tietototyypit käydään lävitse ja hunajasalauksen käytöstä niiden yhteydessä keskustellaan. Viimeisenä keinona gradu tarjoaa toteutusmallin teoreettisella tasolla, mutta mukana on myös käytönnön esimerkkejä tehtynä ohjelmointikielellä. Malli tarjotaan käymällä läpi hunajasalauksen algoritmin runko askel askeleelta. Nämä kolme askelta yhdessä: haasteet, uudet tietotyypit ja toteutusmallin on tarkoitus antaa laajahko näkemys hunajasalauksen tilanteesta koottuna yhteen paikkaan.

Avainsanat: Tietoturva, Salaus, Hunajasalaus, Kryptologia, Väsytyshyökkäys, Hunajasana, Harhautus, Symmetrinen salaus, Salasanapohjainen salaus

Preface

Idea for this thesis originated for my interest for information security, which is also the interest for so many facets such as businesses, governments and individuals. Of course hackers are very fond of it as well, although they are mostly interested in breaking the security than prevailing it. It is all too common to hear about yet another data breach or another attack on some highly known company that it is almost just another news amongst news. This thesis is a small contribution to the field of cyber security, as we need to look to new security methods as the attackers are constantly finding new ways to attack the systems. Let's move from extremely hard to impossible what comes to breaking cryptosystems.

I would like to thank my supervisor Andrei Costin for all the excellent advice and mentoring through the whole process. I would also like to thank Jukka Nieminen, who has been there for me for the past 12 years. Thank you.

Glossary

AES	Advanced Encryption Standard
CCA	Chosen-Ciphertext Attack
CDF	Cumulative Distribution Function
DTE	Distribution-Transforming Encoder
GD	Genomic Data
GHz	Gigahertz, unit of frequenc
GUA	Graphical User Authentication
HDec	Decryption part of Honey Encryption scheme
HE	Honey Encryption
HEnc	Encryption part of Honey Encryption scheme
IoT	Internet of Things
IS	Inverse Sampling
KDF	Key Derivation Function
OTP	One-Time Pad
PBE	Password-Based Encryption
PIN	Personal Identification Number
RSA	Public-key cryptosystem
SHA	Secure Hash Algorithm
VHE	Visual Honey Encryption
XOR	Exclusive or operation

List of Figures

Figure 1. Constructive research method 4
Figure 2. HE modeled with the user typo challenge 36
Figure 3. HEnc framework 39
Figure 4. Key generation with AesManaged library from .Net framework 41
Figure 5. PBKDF2 Key derivation function example 41
Figure 6. Inverse sampling 44
Figure 7. DTE-then encrypt framework 45
Figure 8. HEnc framework 46

Contents

1 INTRODUCTION 1
 1.1 Research question 3
 1.2 Research method 4
 1.3 Data and ethics 4
 1.4 Literature review 5
 1.5 Related work 6

2 CRYPTOGRAPHIC THEORY OVERVIEW 8
 2.1 Related fields 8
 2.2 Information security 10
  2.2.1 Deception 12
  2.2.2 Brute-force attack 14
  2.2.3 Encryption / decryption 18
  2.2.4 Hash functions 21

3 HONEY ENCRYPTION THEORY 24
 3.1 Use cases 24
  3.1.1 Genomic data 25
  3.1.2 Passwords 27
  3.1.3 Password vaults 28
  3.1.4 Multi-dimensional data 30
 3.2 Challenges of Honey Encryption 31
  3.2.1 Production of honey messages 32
  3.2.2 User typos 34

4 HONEY ENCRYPTION FRAMEWORK 37
 4.1 DTE-then encrypt 37
 4.2 Creation of the encryption key 40
 4.3 Inverse table 42
 4.4 Distribution-transforming encoder (DTE) 43
 4.5 Hashing 44
 4.6 Decryption 46

5 CONCLUSIONS AND FUTURE WORK 47

BIBLIOGRAPHY 49

1 Introduction

Software security is a battlefield between the attackers and the defenders. One very common method used by the defenders is encryption. Simply put, encryption is used to ensure that the data stays secured, even if someone else gets hold of that data. Although encryption is a very clever design, it has a major flaw that can be exploited; it can be cracked with a brute-force attack if enough time and resources (e.g. computer power, research advances, money, skilled teams) is given to the attacker. Since computational power is ever increasing, the attackers have easier time to break encrypted data and algorithms. So far, many advancements have been made to prevent brute-force attack, as well as preventing other kinds algorithmic attacks, such as side channel analysis and info leaks. One of the advancements is increasing the length of the encryption key, so it will take longer to break it. This will work for the time being at a decent security level, but for how long? New ideas are needed to secure encryption in the future. Promising ideas have been presented for quantum encryption and for the technique that the rest of this thesis talks about: Honey Encryption.

The term honey has a history of representing something fake in a software security field. Honeypot is used to lure the attacker into a fake target (Cohen, 2006), preventing them for getting information from the real target. The system may have features that allow you to get information about an attacker instead of providing sensitive information to them (Cohen, 2006). Honeyfiles on the other hand are otherwise unused files, and reading them will usually send an alarm (J. Yuill, Zappe, Denning, & Feer, 2004). Honeyword (Bojinov, Bursztein, Boyen, & Boneh, 2010; Juels & Rivest, 2013) is also a fake data, but instead of file it is similar to some other data, such as user passwords (Bojinov et al., 2010; Juels & Rivest, 2013). These honeywords might be stored in the database, and if one of them is used, the alarm is sent, because there is very little probability of an actual user typing these in. Honeytoken was suggested by Barros (2003) as a general term for large range of data that if is basically planted and their usage is monitored. There are some quite recents concepts as well, such as honey chatting (Kim & Yoon, 2016) which uses honey encryption method for securing instant messages. All of these are methods that are meant to be deceitful to a controlled part of the use cases. They share a common idea of using deception as a security method. With that being said, the idea of deceiving the attackers is not locked with the term honey as there are various ideas and methods that are deceitful but do not include the term honey, such as Kamouflage (Bojinov et al., 2010).

In 2014 Juels and Ristenpart published an article "Honey encryption: Security beyond the brute-force bound" (Juels & Ristenpart, 2014b). In that article they introduced the term Honey Encryption (HE) and explained how this new encryption technique could be a viable asset against brute-force attacks, because the attacker could not tell apart the wrongly decrypted data and the correctly decrypted one. This means that even when the attacker tries to use a wrong key, a valid looking result is shown and there is no way of picking the right key just by looking at the results, thus making brute-force attack ineffective. Of course this means that the fake data looks like the real data and can not be recognized as fake.

Because Honey Encryption is a fairly new idea, only some reviews and testing articles are published about it. Even though various proof-of-concept limited implementations exist, at this time there is no known implementations in the production use. This means that the implications of this encryption technique to the large field of data security is hard to predict or evaluate at this moment, but the idea seems interesting and it can provide major improvements for the field of data security. The article written by Juels and Ristenpart (2014b) has been cited in multiple other publications and so it has been influencing new researchers for implementation ideas. For instance, HE has been suggested as a security method with videos and images (Yoon, Kim, Jo, Lee, & Lee, 2015), with human produced messages (Omolara, Jantan, Abiodun, & Poston, 2018), and with genomic data (Naveed et al., 2015; Huang, Ayday, Fellay, Hubaux, & Juels, 2015).

Research has also shown that there are a few challenges associated with implementing Honey Encryption. One major challenge is producing honey messages, i.e., messages that look like a valid result but is actually fake data, used to deceive attackers. This problem has been acknowledged by Juels and Ristenpart (2014b) as *"A key challenge for HE"*, as it may be especially difficult with data that consist of natural language. Another challenge that the researchers have highlighted is the typos made by the users when creating real data (Juels & Ristenpart, 2014b). If the system utilizes users input, e.g. with passwords, there is always the possibility of a misspelling the word. If the Honey Encryption returns fake data when the input is incorrect, wouldn't the typo also return fake result? It is obvious that this kind of scenario should be handled in a different way than a response-to-attack situation.

In order to provide some information about those challenges, this thesis does an exploratory and experimental research on the challenges of Honey Encryption. It explains what Honey Encryption is, what known challenges it has and what has been suggested as solution for those challenges. All of this is done with a systematic literature review, as those models presented in the thesis weighs heavily on the research results found in the exploratory research.

The rest of this thesis is organized as follows. Chapter 1 will explain research questions and research methods. Chapter 2 discuss the related fields and important concepts in cryptography. Chapter 3 will dive into Honey Encryption theory and Chapter 4 provides a model of the framework and architecture of Honey Encryption. These chapters together will offer both theoretical and practical information about HE and therefore provide very comprehensive understanding of Honey Encryptions current state. Last chapters will offer some ideas for future research and the Chapter 5 will discuss the result found as a result of this thesis work.

1.1 Research question

When HE was introduced, there were some challenges mentioned in the article itself (Juels & Ristenpart, 2014b) and those challenges are still being under research. This thesis gathers those challenges together into one place as an inspiration for future research. It is important to gather the current knowledge about HE into one place and provide a step by step implementation model so that the future work can go on and actual implementations of HE can be used in practical field as well. This brings both scientific and practical value to the thesis.

The research questions that are broadly addressed in the thesis are:

- RQ_1 : What challenges Honey Encryption brings into the research field?
- RQ_2 : What feasible ideas/tools exist towards working Honey Encryption implementation in the real-world system?

1.2 Research method

Inspired by the research questions, a constructive research model (Lukka, 2003), is used as a research method for this thesis. Figure 1) is modeled after Lukka's explanations of the 7 steps for the research method.

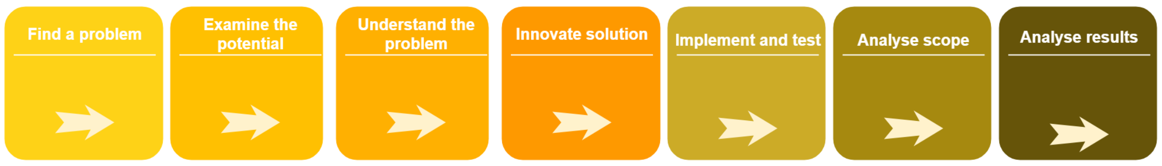

Figure 1: Constructive research method

As Lukka (2003) explains, the objective for this research method is in *"producing innovative constructions, intended to solve problems faced in the real world and by that means, to make a contribution to the theory."* The first 3 steps are done with a systematic research review and the problems are represented as research questions. Examining and understanding the problem is presented in the theory part in the thesis, mainly in chapters 2 and 3. In the 4th step, innovating a solution, it is important to provide some new research information to the field instead of just presenting something in a new way or simply copying the current knowledge (Lukka, 2003). This part is always a bit risky, since it might be that there is not anything new to discover and therefore there is no innovation. In this thesis the innovation phase contains gathering multiple ideas and results together and provide a clear model for the current state of HE. The 5th step is implementation and testing. In this thesis the testing has mostly already been done, as the results rest on previous results. Finally the results are analysed and discussed in the discussion chapter.

1.3 Data and ethics

Data will consist of literature gathered in the exploratory research. Research data will not contain people's personal information and therefore Privacy notice template is not needed in this research (University of Jyväskylä, 2019). This paper will be inline with the values and good practices that are listed by the Finnish national board of research integrity TENK (Finnish Advisory Board on Research Integrity, 2012). Any concerns about ethics in data security and moral dilemmas, e.g. what kind of data should be kept private, is out of the scope of this thesis.

1.4 Literature review

It is an ideal time to do research of Honey Encryption. The idea of HE is fairly new, so it is current and there are multiple research ideas left unanswered. At the same time, HE was presented years ago, so it is possible to perform a decent and representative literature search about it. The goal of the literature review is to find all relevant references to honey encryption and to the scientific fields behind it. Predicted challenges included lack of sources as it is quite a new and explicit idea.

Main source of the literature review is the original article which presented the idea of Honey Encryption. It was presented by Juels and Ristenpart in 2014 and it is called *"Honey Encryption: Security beyond the brute-force bound"* (Juels & Ristenpart, 2014b). By using that article, all references to it were looked through. Google scholar found 63 references to that article at 11.3.2019. Four of those were written in a foreign language and for that reason they were discarded. This left 59 references to the original article. Out of those 59, 17 contained relevant information in regarding of research questions and were added as references for this thesis. The other 42 were discarded as most of them discussed honey encryption as related work or merely mentioned it. Some of those papers offers interesting references themselves and those references were included as well.

The reference list from Juels and Ristenpart (2014b) was also looked through as those sources could contain important background information. Earlier work from the writers Juels and Ristenpart (2014b) was also looked through and relevant items were included. Two targeted searches was made: with the word "security deception honey" and with "cryptography". The search engine for both of those queries was google scholar. Other sources include more practical references that has less scientific value, but offer interesting real-world data. These types of references include blog posts and public programming code libraries. Three references were included from universities thesis lectures and they contained the information for research methods and data handling (Lukka, 2003; Isomäki, 2018; Finnish Advisory Board on Research Integrity, 2012).

It was noted after the literature review was finished that there was more research done about HE than was predicted in the beginning. Even though a lot of the papers that referenced the original HE article simply mentioned it by related work and did not add anything new to it otherwise, there were a lot of papers that did go beyond the current knowledge of HE and offered new ideas or offered reviews about it. This made the literature review more successful than originally planned.

1.5 Related work

Honey Encryption embodies some familiar features from other systems. Especially systems that uses decoys (Cohen, 2006) as part of the defence. Perhaps the most famous security decoy idea is honeypot (Spitzner, 2001; Cohen, 2006), where one lures the attacker in with some interesting, but fake information. Cohen (2006) quotes a Jesus Torres for saying *"For a honeypot to work, it needs to have some honey"*. It this sense the honey refers to the attractiveness of the fake data and the ability to get attackers to target the honeypots. HE system itself is not trying to lure attackers in, so in that sense HE does not have "any honey", but it it still part of the same deception genre as the honeypot.

HE scheme contains the use of Honeywords (Bojinov et al., 2010; Juels & Rivest, 2013), which are the fake message that HE will produce when necessary. Honeywords are not solely part of HE, as they did exist as a concept before HE came along. Because honeywords are not solely connected to HE, the research done about honeywords will probably have an independent research line. As an example, in 2018 there was quite a comprehensive review done about honeywords security (D. Wang, Cheng, Wang, Yan, & Huang, 2018). Because research that investigates honeywords may contain useful ideas about honeyword creation, this research line may intervene with HE and help with the challenge of producing valid honeywords.

Special mention must go to Kamouflage system, a method used to prevent offline attacks on PBE systems by storing fake passwords alongside user actual password (Bojinov et al., 2010). It was especially targeted for password vaults as the fake messages produced in the

Kamouflage system contained sets of passwords. It is easy to see the resemblance between Kamouflage and HE since they both have the same idea of deceiving the attackers and preventing them from recognizing the correct data (e.g., user real password) from intentionally wrong data (e.g., fake password sets).

At the same time as Honey Encryption was presented, another system called structural code scheme was also published (Jo & Yoon, 2014). It was also a method against brute-force attack and it used, similarly to honey encryption, the idea that every key can be used and they would yield plaintexts that would deceit the attacker (Jo & Yoon, 2014). However, structural code scheme lacks formal mathematical proof (Jo & Yoon, 2015), but offers syntactic security and the ability to use natural language (Jo & Yoon, 2015). Structural code needs more research done, just like Honey Encryption, in order to provide guaranteed and verifiable security in the field.

2 Cryptographic theory overview

This chapter will explain the basics of modern cryptography, focusing on topics that are related to honey encryption. This should offer the essential information needed for understanding the context behind honey encryption.

2.1 Related fields

Cryptology. Oppliger (2011, p.1) explains that cryptology as a term is combination of two terms: Crypto and logy. The term Crypto has a Greek meaning of "Hidden" when traced back from the Greek word *"kryptos"*. Logy on the other hand can be traced back to the word *"logo"*, which has a meaning of *"word"*. This way the term Cryptology can be interpreted as *"hidden word"*. (Oppliger, 2011). Cryptology consist of cryptography and cryptanalysis (Van Leeuwen & Leeuwen, 1990, p.720). This means it is a study about protecting and analysing/breaking cryptosystems.

Cryptography Cryptography is a subcategory for cryptology and it has the same first part as cryptology, but the second word "graphy" has the meaning of "write" (Oppliger, 2011). This way the interpreted version of the term is *"hidden write"* or as Oppliger (2011) says in a better way: *"hidden writing"*. Van Leeuwen and Leeuwen (1990, p.719) has presented short and precise explanation of it: *"Cryptography is about communication in the presence of adversaries"*. One of the main goals of cryptography is confidentiality (Katz, Menezes, Van Oorschot, & Vanstone, 1996; Diffie & Hellman, 1976; Van Leeuwen & Leeuwen, 1990, p. 719, written by Rivest) although Rivest, and Diffie & Hellmand did use the term *"privacy"* instead of confidentiality.

Other goals in cryptography are authentication (Diffie & Hellman, 1976; Katz et al., 1996; Van Leeuwen & Leeuwen, 1990, p.719), Data integrity and Non-repudiation (Katz et al., 1996). Authentication has overall meaning of making sure that the only the appropriate people can view or modify data (Katz et al., 1996). Data integrity cannot be achieved with encryption alone (Katz et al., 1996), therefore honey encryption does not guarantee it either. It guarantees that the data is as it should be and it has not been altered by an attacker for example (Katz et al., 1996). For example, is a person could alter the messages in the messagespace, the encryption process would still work, but the data integrity would be broken. Non-repudiation will make sure that actions cannot be deleted from the history (Katz et al., 1996).

Even though cryptography has a long history, it has an important part of modern software security as well, because it is still used for protecting information (Martin, 2012, p. 2-3). The honey encryption algorithm is part of cryptography, because *"The science of constructing encryption algorithms and related systems is known as cryptography"* (Bidgoli, 2006, p.469) The work of Juels and Ristenpart (2014b) of developing the idea of honey encryption, and even this thesis is working towards better scientific and practical knowledge of cryptography.

Cryptanalysis is a subcategory for Cryptology and it consist of two words, where the second part *"analysis"* can be interpreted as *"to loosen"* according to Oppliger. From this Oppliger presents the meaning of *"loosen the hidden word"*. (Oppliger, 2011). Cryptanalysis is trying to figure out ways to break cryptosystems (Oppliger, 2011; Katz et al., 1996; Van Leeuwen & Leeuwen, 1990, p. 720) with the use of mathematical techniques (Katz et al., 1996). This thesis includes some work that lies within the general direction of cryptanalysis in the form of looking up known information of honey encryption problems. This thesis does not include actual own analysis of Honey Encryption weaknesses. Even though objectivity is a goal for this research, the aim for the researcher is probably to find the Honey Encryption technique effective, and therefore the cryptanalysis part of the research must be considered as not complete and little bit biased.

Cryptographic system or cryptosystem is a *"general term referring to a set of cryptographic primitives used to provide information security devices"* (Katz et al., 1996). Rivest explains that a cryptosystem is a scheme that is working in benefit for cryptography, trying to keep the communication secured (Van Leeuwen & Leeuwen, 1990, p.720). This securing of communication can be presented with a mathematical formula (Diffie & Hellman, 1976):

$$S_k : \{P\} \rightarrow \{C\} \tag{1}$$

where P is possible plaintexts, C is ciphertext, K is cryptographic key and S is encryption/decryption (Diffie & Hellman, 1976). This explanation does not exclude adversaries tools however, if those tools has their own cryptographic intentions as well. An adversary may want to have the same confidentiality, even if the ultimate goal is to break someone else's confidentiality.

Social engineering may interest people who are into the deceitful part of Honey Encryption. It is not however discussed in this thesis any further, because it uses human interaction as part of the attack. These interactions include person deceiving a person in order to proceed with the attack (Mitnick & Simon, 2011). Example of this would be pretending to be a worker and accessing private computer in that company.

2.2 Information security

Honey Encryption is an example of previously explained cryptosystems. Security can not be emphasized enough when it comes to cryptosystems such as HE, as their goal is to secure data and increase security itself. Poorly implemented security systems can even lower the overall security by creating false sense of security and therefore prevent the usage of other security systems instead. It could even set up a motion of turning off other security methods or create false data to administrators, who would make misguided actions based on that information. For example, the system could send unnecessary alarms or not notice an intruder in the system. If secure implementation of HE is ever planned to be done, the definition of security must be done as well.

System that uses HE will most likely use other security components as well. This makes HE merely an information security service (Katz et al., 1996, p. 14) in a larger cryptographic system that is protecting even larger software and data system. Without the larger software system HE would not have any purpose for storing the data. Who is it for? Data that would exist for the mere purpose of HE is educational or for learning purpose only, not for the real-life applications. Those kinds of data storages may be important part for research, but they are not the end goal. For this thesis however, only those elements that are strictly connected to Honey Encryption, are discussed and these kinds of naive data storages are used as an example. It is worth mentioning nonetheless, that you cannot separate security from its real life application. With the words of Schneier (2011) *"Cryptography doesn't exist in a vacuum"*. Therefore, even a perfect implementation of HE does not guarantee a secure system in itself.

So how do we guarantee the security? In overall, the information security consist of the following goals: authentication, data integrity and confidentiality (Katz et al., 1996, p. 14). At this time Katz used the term privacy as a synonym of confidentiality and that is the most important goal for HE, because confidentiality means keeping the data secured from unauthorized changing (Katz et al., 1996, p. 14). As an example, instead of attacking the database, the attacker might change the data inside it. The prevention of this kind of unauthorized changing of data goes inline with the goals of HE, as it is trying to prevent brute-force attacks and so make it unaccessible to anyone but the correct entities. This would keep the attackers out but let the correct entities in.

Authentication and data integrity can be part of HE, depending on the system requirements. Authentication is making sure that the person, or what ever entity is accessing the data, is really the correct entity (Katz et al., 1996, p. 14). For example, if HE is used with passwords-based encryption, it is in fact part of the authentication process, because the user is authenticated with a password that is encrypted with the HE scheme. The two processes: authentication and confidentiality are however distinct (Katz et al., 1996, p. 14) and mean different things.

Last element in the security goals was data integrity. Katz et al. (1996, p. 14) mentions that data integrity is a mechanism to protect against unauthorized manipulation of the data, including deletion, addition and substitution. This substitution is exactly what HE will do if it notices an alarming situation. HE will deliberately break the data integrity by providing false data if incorrect key is used as part of the decryption process. For example, if HE is part of logging in to password vaults, and the alarm is raised for using an incorrect password, the system could provide the user with a fake password vault information. Even though this substitution is in the core of HE scheme, and the fake result are the key concept of its implementation, it is done for ensuring the confidentiality. Therefore, data integrity in itself is not the key goal of HE, but merely a method for securing the confidentiality of the data.

Schneier explains that security is a process involving mathematics, people and computers, and you cannot create security with just mathematics. (Schneier, 2011, p. 7). Schneirer was referring to the system as a whole, where you need to take into consideration all aspects of the system as well as their connections and how they are interacting. Even though HE can be constructed as a individual component, that runs mostly without a human interaction, it is not completely free of the human touch. As Schneier (2011) mentioned: *"Security is a chain; it's only as secure as the weakest link"* (Schneier, 2011, p. 8). Humans can mess the HE process if developers create unsafe products or administrators leak the data or modify alarm settings. If the HE has an alarm process, as it should have, that alarm process may need some administrative status people to take certain actions. These kind of human-related security factors are left out of the security scope for this thesis as they fall more on the social engineering side of security.

Defining data security is a broad topic, and impossible to cover in this thesis. One definition was made in effort to have somewhat clear understanding of the generic security goals of the HE system. Data security is summarized as: *a process to guarantee authentication, integrity and confidentiality of data in a cryptographic system, with the use of people, mathematics and computers* (Schneier, 2011; Katz et al., 1996).

2.2.1 Deception

Deception has been a security method in many systems, such as Kamouflage (Bojinov et al., 2010) and Honeypots (Spitzner, 2001). Perhaps one of the earliest mentions of deception used as a method against hackers has been done by Stoll (1989, p. 41-42) and Cheswick (1992) when he describes a fake data being planted in order to mislead a hacker. Earlier we defined HE as a information security service according to the definition made by Katz et al. (1996, p. 14). Here we make that definition more accurate by defining HE as deceitful security service. But first we need to discuss about what that means.

J. J. Yuill et al. (2007) explains deceptions in a following way: *"Computer-security deception is defined as the planned actions taken to mislead hackers and to thereby cause them to take (or not take) specific actions that aid computer security defenses"*. J. J. Yuill et al. (2007) will further make a distinction between denying information and deception, as he will explain that deception always have the intent to mislead, whereas denying simply does not give access to the data (J. J. Yuill et al., 2007). This would make HE service definitely a misleading method, because it is not meant to deny access. Although it can be used with a larger cryptosystem that, in some specific cases, will deny access based on certain factors that are connected to Honey Encryption.

This definition leaves out the perception of the attacker may have. Does he need to be unaware of the deception? At least in theory there is a possibility that the attacker knows that HE is being used. The attacker might gain this knowledge simply figuring it out through his attacking scenarios or the system might be open source project. Is deception a deception if the attacker knows about it? According to J. J. Yuill et al. (2007) it is, because if we take another look at his explanation, when we design and implement HE, that is the *"planned action"* Yuill was talking about and that action is meant to mislead the attacker. In a simpler terms, knowing that the system uses HE and decryption returns false data does not transform it to a valid data and the deception stands.

Even if we take out the part of HE where the decryption process return fake data, the encryption process itself can be considered as a deceitful method on its own (Cohen, 2006). Sometimes encryption is not specifically listed as the deceitful method (Almeshekah & Spafford, 2014). If we take a look at the definition by J. J. Yuill et al. (2007), encryption can easily be classified as planned actions to misguide the attacker. For example, in PBE the encryption or hashing is not necessary for the logging in. It is done purely as a security method, in case the confidentiality has been broken. Almeshekah and Spafford (2014) uses term **repackaging** for methods that are making data appear to be something else. This could fit into the encryption concept quite well and make HE scheme deceitful in multiple ways, the conventional encryption as one part and the fake messages as another.

Han, Kheir, and Balzarotti (2018) would also say HE is a deceitful method as they listed HE as one of the recent deceitful techniques that they evaluated. Interestingly only 15% of the surveyed techniques were preventive methods such as HE . The idea of preventive method was explained as a method that take actions to prevent the attack before it happens (Han et al., 2018). Preventive nature of HE would most likely come from the ability to withstand brute-force attacks. Overall the ability to prevent data leaks is obviously important and this makes HE very useful security method.

Han et al. (2018) also discuss two other deception mechanisms: detection and mitigation. The idea of detecting if data has been compromised has existed before the idea of HE. One trick has been setting up fake accounts and send the alarm if those accounts are used (Juels & Rivest, 2013). Even though HE is listed as preventive deception mechanism in the paper, the overall HE system can use all three types of deception techniques: prevention, detection, and mitigation, depending of the overall framework. If the system uses honeywords, as the original HE scheme does, any usage of honeywords will send an alarm. The usage of alarms is definitely more detective than preventive action.

Mitigation on the other hand is defined by Han et al. (2018) as an active effort for disrupt the ongoing attack, such as using a decoy system. This is exactly what is used in the HE model in this thesis. If the system notices an attacker in the system, they could possibly use a honeypot version of the system. Therefore it is quite accurate to say the HE scheme is *deceitful technique that can utilize mitigation, detection, and prevention as a method to secure the system* (Han et al., 2018) and even the most compact version of HE, without any alarms or honeypots is at least preventive.

This section defined deception and explained that it is quite a broad topic, just like data security is. Very generic definition for data security made in section 2.2 is not enough to define HE, as it is too abstract. For that reason another definition is made for the HE itself that includes the idea of deception: *Honey Encryption is information security service, that break data integrity in the purpose of misleading the attacker and to guarantee confidentiality of data in a cryptographic system, with the use of mathematics and computers* (Schneier, 2011; Katz et al., 1996). The idea of human interaction is left out and the deception part is added from the previous definition made in Section 2.2.

2.2.2 Brute-force attack

This section describes the most important attack type that relates to HE system: brute-force attack. It is the most important attack type in thos context because HE is promised to provide security against it (Juels & Ristenpart, 2014b), and brute-force attacks have been bothering encryption techniques and caused data leaks since encryption was invented. If there is a new method to prevent brute-force attack, such as HE, it would be quite a important step forward. There are two kinds of brute-force attacks, offline and online versions and both of these are included in this section.

One way to attack a cryptosystem is trying every possible decryption key. This attack type is the infamous brute-force attack (Oppliger, 2011). Oppliger (2011) explains that the average time it takes to find the correct key is

$$\frac{|K|t}{2p} \tag{2}$$

where $|K|$ is the number of possible keys (*the key space*), t is time to use one key, p is how many processors there are. The time is divided by two, because it takes about half the time on average to find the right solution (Oppliger, 2011).

It is quite obvious it takes a lot of resources from the attacker to perform full brute-force attack. This is why some attackers do not try every possible item there might be, they don't exhaust the whole key space, but rather they try on smaller part of the possible keys. Problem is, if you are going to try on only a part of the possibilities, which ones to choose? Randomness is not as efficient as intelligent guesses. Intelligent guesses on the other hand depends on the data itself. If there are e.g. more common values, then you want to give those values more statistical probability to be tried on as a key. One can notice how this would be a problem with password-based encryption, where a portion of users tend to choose weak passwords (Taneski, Heričko, & Brumen, 2014). When data leaks happen for passwords and those passwords lists become public, they provide a neat list for intelligent guesses in a gold plate. In 2010 this kind of leak happened in China, known as s CSDN Password Leakage Incident (Zhigong Li, Han, & Xu, 2014), where over 100 million passwords was leaked in plaintext for the public to download. The passwords came from five different sites that were hacked in a matter of days.

Of course methods such as hashing and salting the password provides enough security so that the attackers resources are exhausted before the key space is. However, the resources that the attacker is able to use are growing and it takes less and less time to provide enough computational power for full brute-force attack. This is the reason why the enryption keys or the outputs of hash functions are growing. For example, MD5, which has been found insecure (Dougherty, 2009; VU, 2014; Sotirov et al., 2008), has 128-bit output. SHA-3 has the minimum of 224 bits. The comparison of the bit array length alone is not however a good measure of algorithms, but serves as a one factor.

Growing the key length may not be feasible tool in the long run and therefore another solutions have been looked for quite some time. Especially alternative methods for passwords has been research topic for some time (Bonneau, Herley, Van Oorschot, & Stajano, 2012). HE is a good candidate for extra security in PBE scheme, but also for other kinds of data against brute-force attack. These data types are discussed in Section 3.1.

HE provides security against brute-force attack by offering valid-looking plaintext every time the ciphertext is decrypted (Juels & Ristenpart, 2014b). This way the attacker is unable to know whether he has the correct key. Juels and Ristenpart (2014b) explains that this makes HE scheme computationally secure, because even if the attacker has unlimited computation power, he will not be able to break the system, or even if the attacker breaks it, he or she might not know it because every message looks the valid ones. How to know which one to use as the password?

In order to have a successful attack of this type, the attacker must be able to distinguish correct plaintext from the incorrect one (Oppliger, 2011, p. 132; Juels & Ristenpart, 2014b). In a normal situation, as Oppliger tells, is that there may be multiple solutions, that looks like correct plaintexts. There will be a lot of results that do not look like valid results, and the attacker can rule those results out (Abiodun & Jantan, 2019). Juels and Ristenpart uses the term brute-force bound to describe the issue of guessing the correct data (Juels & Ristenpart, 2014b). That means that even if he has all the resources to perform full brute-force attack, he will not know which message was the right one. The attacker can make a guess, but if HE is done properly, the probability guessing the right key is as good as guessing any of the keys

$$1/\{K\} \tag{3}$$

where {K} represents the key space.

Offline brute-force attack can happen when data breach has compromised encrypted data (Harsha & Blocki, 2018). After that the attacker is free to try as many keys as he has resources to. *"Offline brute-force cracking attacks are increasingly dangerous as password cracking hardware continues to improve and as many users continue to select low-entropy passwords"* (Harsha & Blocki, 2018). This is important difference with the online brute-force attack, where there are usually a limit of the guesses that can be done. After the limit is met, the account can be frozen. The logging attempts can easily be monitored as well.

Juels & Ristenpart explains that in message recovery attack, the attacker is in possession of the decrypted message. He will try to figure out what the plaintext was, the one used with the key in order to be encrypted. Problem is, that the attacker does not know the decryption key and have to therefore make guesses. In other words, he is trying to decrypt the message without knowing the right key (Juels & Ristenpart, 2014b).

To be accurate, there already is a method that has been proven to be secure against brute-force attack: one time pad (OTP) (Gisin, Ribordy, Tittel, & Zbinden, 2002). So in a way HE is not the first technique to provide security against brute-force, but it does is with decent amount of resources, at least what comes to the encryption key. One time pad uses an encryption key that is as long as the plaintext is (Aumasson, 2017; Van Leeuwen & Leeuwen, 1990, p. 720). The idea of one-time pad was invented in 1926 (Gisin et al., 2002) and it has been in actual use during the 1960s (Cohen, 2006). Aumasson (2017) explains that one time pad simply XORs the plaintext with the encryption key, where P is plaintext, K is the key and $\oplus$ is XOR -operation. (Aumasson, 2017):

$$C = P \oplus K \tag{4}$$

If the key is as long the plaintext, as it is in OTP, the intruder is unable to exclude anything out of the results. However, Aumasson explains that the use of OTP may not be the best idea in every situation, because of the large amount of memory it takes to store the key. Data storage needs to be doubled, because the encryption key takes as much space as the data itself (Aumasson, 2017). Maybe for this reason Rivest called this method *"nearly perfect cryptographic solution"* (Van Leeuwen & Leeuwen, 1990, p. 720).

Brute-force attacks, offline brute-force attacks and message recovery attacks are basically trying to do the same thing: they are trying to get a hold on to the original plaintext. They are the real concern of any system that uses encryption techniques, unless the system is OTP or some other method that contains security against them, such as HE.

2.2.3 Encryption / decryption

HE contains a new scheme for the encryption process called DTE-then-encrypt (Juels & Ristenpart, 2014b). For that reason it is important to define the encryption process in general and the key components that it consist of.

Encryption is transforming the message in a way that eavesdropper is not able to understand it, even if he got hold of the message (Cohen, 2006). Aumasson explains that in encryption you need five parts: ciphertext C, plaintext P, key K and the cipher that is responsible for the encryption E and decryption D functions.

$$C = E(K,P) \tag{5}$$

$$P = D(K,C) \tag{6}$$

The ciphersuite is doing encryption or the decryption each time it operates. It is either transforming plaintext into ciphertext or ciphertext into plaintext. Both of them use a key in the process. If symmetric encryption is used, the same encryption key is used in both processes (Aumasson, 2017) otherwise, in asymmetric schemes the public key is used for encryption and the private key is used for decryption.

Kerckhoffs principle says that security cannot rely on the fact that the attacker won't know details of the cryptosystem (Oppliger, 2011, p.1). For example the security of encryption should rely on secure key (Cohen, 2006). In regards of honey encryption we shouldn't rely on the fact that the attacker does not know that honey encryption exist or is in use in our cryptosystem. An attacker therefore will know that honey messages are produced and the reason for doing that is to deceit the attacker. Because attacker is presumed to know about deceitful honey messages, he won't be deceived by merely seeing them. This will not matter, because the attacker has to be not able to recognise the honey messages from the real one. This way the attacker will not benefit for merely knowing about honey messages as he can not access the system anyways and Kerckhoffs principle will hold.

Key space is important part of encryption and the one factor that has grown as new encryption algorithms are developed. Overall the key space follows the formula of 2^n where n is the amount of bits used for the encryption key. E.g if n = 3, the possible keys could be:

000	001	010	100
011	101	110	111

The probability of guessing the correct key can be calculated as $1/2^n$ (Aumasson, 2017), where n is the bit amount. This makes the probability of guessing the incorrect key $1 - 1/2^n$ and in the previous example that would be $pr[1/2^3] = 0.125$. Now, if the encryption key would be 56-bits, like used in DES (Aumasson, 2017) the probability of guessing the correct key rises to

$$1/2^{56} \tag{7}$$

and with 256 bits as used in AES-256, the probability rises to

$$1/2^{256} \tag{8}$$

Which is extremely close to 0. So the probability for picking the incorrect key at random is close to 1. It is not this probability we are after however, because the attacker does not need to pick the correct key in the first try. In brute-force attack he can try on every single one of them and eventually he will get the correct one. Probability for getting the correct key if the attacker goes through all possibilities is simply 1, and that is what is worrisome. In order to prevent this from happening, they key space has to be big enough so that it takes too much resource and time to go through them all. So what we are really interested when evaluating brute-force attack is the resources, not the probability, although the probability for pickin the correct key has to pretty low as well.

The time needed to break the encryption with just 3-bit key would not take that long. Using the algorithm presented in section 2.2.2, the formula would be: $\frac{|K|t}{2p} = \frac{8t}{2}$, as simplification reasons we have only one processor so p is 1. So how long does it take to perform a single decryption? What is the t? In many cases it has been suggested that one operation takes one billionth of a second, equivalent to nanosecond, to perform (Aumasson, 2017) An example basic home computer in this case could have 4.3 GHz. This means it can roughly make a 4,3 billion basic/elementary operations in a nanosecond, which is equivalent to roughly 4 billion billion operations in a second. It is difficult to estimate how many of these single operations is takes to validate a single key, and here we use the same estimation as Aumasson (2017) that is takes 1 nanosecond. Therefore t is 1 nanosecond.

In asymmetric encryption, also known as public-key encryption, a different key for encryption and decryption is used (Gisin et al., 2002; Aumasson, 2017). Attacker will be mainly interested for the decryption key, because that will enable him to get access of confidential data. Honey Encryption however has similar functioning as the conventional symmetric encryption (Juels & Ristenpart, 2014b) and therefore symmetric encryption is discussed more from now on.

Symmetric encryption was developed earlier than asymmetric encryption and it uses the same key to both encrypt and decrypt data (Gisin et al., 2002). The basic idea has been presented by Gisin et al. (2002), where m is the message, $\oplus$ is addition modulo 2 and C is the ciphertext.

$$m \oplus k \rightarrow C \tag{9}$$

Symmetric encryption needs a secure channel in order to change the keys with both sides, so that the message can be decrypted (Katz et al., 1996; Gisin et al., 2002). This problem is known by key distribution problem (Katz et al., 1996).

2.2.4 Hash functions

HE uses conventional hashing as part of DTE-then encrypt process (Juels & Ristenpart, 2014a), and therefore it is important to explain the basic functionality for cryptographic hash functions. Hash function is designed in a way that the encryption is made easy, but decryption is made difficult. This is because the hash function maps the set or inconsistent length bit strings into a fixed length bit string (Katz et al., 1996, p. 33). This makes these functions one-way functions (Katz et al., 1996, p. 33). Examples of hash functions are MD5, SHA-1, SHA-2 and SHA-3. Currently MD5 has been found as insecure (Dougherty, 2009; VU, 2014; Sotirov et al., 2008) and there has been found collisions from SHA-1 (Stevens, Bursztein, Karpman, Albertini, & Markov, 2017). Collision means that multiple values will end up having the same result.

Hash functions h, takes a value v in and produces a hash result r (Katz et al., 1996, p.231).

$$h(v) \to r \tag{10}$$

Hash result is always the same length, no matter what size the value was, so the hash size is fixed. (Katz et al., 1996, p.231). For example SHA256 will produce results that are 256 bits. This 256 bits can be presented as 32 bytes. Here is an example of hash function result for the string "cat", using SHA256 algorithm. Each byte is presented in hexadecimal format, so each byte is presented with two characters.

$$"cat" \to 77af778b51abd4a3c51c5ddd97204a9c3ae614ebccb75a606c3b6865aed6744e$$

The previous hash is formed by the following C# programming code, followed by guide (Microsoft Docs, n.d.).

```
HashAlgorithm algorithm = SHA256.Create();
var hash = algorithm.ComputeHash(Encoding.UTF8.GetBytes(input));
```

It is highly recommended to use salt S, in the hashing process H (Juels & Ristenpart, 2014a; Kaliski, 2000). It is especially important for messages with low-entropy, such as passwords. Salting reduces the risk that the password can be found from rainbow tables. For example, the salt can be some variable length bit string, presented here as "randomsalt". That value is the added to the original pasword "cat" and the new value is therefore: "catrandomsalt". This combination of the password P and the salt S is then hashed with the hash function h.

$$h(P,s) \tag{11}$$

When the user is trying to log in, he will type the password: "cat". The salt is always added to the password, and that hash is compared to the one in the storage. Obviously the salt can be used numerous kind of different ways and the one presented here works simply as an example.

Previously there was a mention that passwords have a low-entropy. In the study done in 2007, there was a discovery that passwords chosen by users contained about 40 bits of entropy (Florencio & Herley, 2007), which is equal to having 40 random bits. If that 40 bits is divided into bytes, that is only 5 characters. This means that the average possibility for each message is somewhat high. Entropy in this scenario basically the certainty of the passwords. The mathematical formula was presented by Shannon (1948), where K is a positive number, probability p_i:

$$H = -K \sum_{i=1}^{n} p_i \, log \, p_i \tag{12}$$

HE uses conventional hash as part of DTE and Juels and Ristenpart (2014a) recommends the use of a random value for each plaintext P. This way the salt is unique for each message that is hashed. The salt could be based on some factor on the user, or even a random salt. However, the salt needs to be stored somewhere because it is used is the decryption process.

Juels and Ristenpart (2014a) also recommend using Key Derivation Function (KDF). The idea of the KDF is to produce another key from a key (Moriarty, Kaliski, & Rusch, 2017), although Moriarty et al. (2017) points out, in a password based encryption process, the original key can be the password. In the RSA standard version 2.1, there is two KDFs explained: PBKDF1 and PBKDF2 (Moriarty et al., 2017). When designing a HE implementation, the recommendation is to use PBKDF2 if possible, because it is for new applications and provides better security Moriarty et al. (2017).

3 Honey encryption theory

The previous chapter explained important background information about cryptography and provided definitions for security, encryption and explained brute-force attack scenario. This chapter will explain the Honey Encryption (HE) itself and it starts by explaining the possible use cases for the HE scheme. After that the main challenges that HE is facing is discussed. There are some important concepts related to HE and these will be explained first to make the rest of the theory understandable.

Because the terminology can be confusing at times, the following definitions are made:

- **Honeyword**. Fake message. (Juels & Rivest, 2013). Other terms: *Honey message* (Juels & Ristenpart, 2014b), *chaff, decoy, incorrect password* (Juels & Rivest, 2013).
- **Sweetwords**. A set that contains both incorrect message and the correct one. (Juels & Rivest, 2013). Another terms: *potential passphrase, potential passwords, sweetphrase* (Juels & Rivest, 2013). If the term is used as a unit *sweetword* it refers to a message, that can be either sugarword or a honeyword.
- **Sugarword**. The correct message. The one that is not fake. (Juels & Rivest, 2013)
- **Honeychecker**. Service that stores the index of the sugarword (Juels & Rivest, 2013)

3.1 Use cases

Juels and Ristenpart gave various examples of different types of implementations of HE, where they used it for Personal Identification Numbers (PIN), credit card numbers, credit card verification values and RSA secret keys. (Juels & Ristenpart, 2014b). These were provided as examples and as a proof-of-concepts and they mentioned that: *"HE is applicable to any distribution of low min-entropy keys, including passwords, PINs, biometrically extracted keys, etc. It can also serve usefully as a hedge against partial compromise of high min-entropy keys"* (Juels & Rivest, 2013).

Additionally to the original examples, researchers have explored other data types and systems that could take advantage of the HE scheme. Some of them contained a additional data type, that was not mentioned in the original paper by Juels and Ristenpart (2014b), such as genomic data, and videos and pictures. Some research papers discussed more about the application type, such as including IoT-devices (Choi, Nam, & Hur, 2017) and public cloud storages (Mok, Samsudin, & Tan, 2017).

3.1.1 Genomic data

One interesting field for HE seems to be storing Genomic Data (GD). Security itself plays especially important role with genomic data, due to the fact that it contains *"valuable and sensitive information about individuals"* and any data exposure can cause various problems, including e.g., blackmail or discrimination (Huang et al., 2015).

Multiple papers have addressed the need for securing genomic data in secure way, especially because the long-lasting nature of it (Naveed et al., 2015; Huang et al., 2015; Humbert, Ayday, Hubaux, & Telenti, 2013). Individuals genomic data should be secured up to persons lifetime and even longer (Huang et al., 2015; Humbert et al., 2013). One paper explained that even partial genomic data can be used with statistical methods in order to gain the part that was left out, e.g. genes in DNA (Humbert et al., 2013) and even after the person have deceased, it can be used to gain information about offspring or relatives (Humbert et al., 2013). Even if person himself does not donate his DNA to any library, he can be identified via DNA samples, if relatives of that person have done so. This is how the golden state killer was finally discovered, although the journey took 47 years and breakthroughs in genomics and search match algorithms. The killer left some DNA in the murder scene and the investigators looked it up from an online storage for genomic data, called GEDmatch (Kolata & Murphy, 2018).

It is not only the long lasting nature what makes genomic data different than the rest of medical data. We do not yet know everything there is to know about genomes. As Naveed et al. (2015) points out: *"Every day, we learn something new about the genome, whether it be knowledge of a new association with a particular disease or proof against a previously reported association"*. Before we have clearer view of all of the information that can be obtained from genomes, it is important to keep them secured. This means that the data should not only be secured, but be secured for a long time.

Genomic data is also largely different than any other data, Naveed et al. (2015) goes as far as calling it special, because there is not other data type that is quite like it. In general sense, it is very unique, because every individual has different genomes and it does not change too much over time (Naveed et al., 2015). Fingerprints and iris are also unique to each person, but those can be modified with surgical operations, unlike genomes.

Genomic sequencing was estimated to cost about 1k$ and last about a day to computationally construct (Naveed et al., 2015), so it is not easy nor completely cheap. One sequence has billions of base pairs (Naveed et al., 2015), which is probably the reason for why it has been difficult. However, it is not the time and costs of the construction for why they should be kept private, as those are mentioned just for the understanding the difference e.g. with passwords of other data types mentioned in this thesis.

Genomic data is discussed in this thesis so widely because it is important in so many fields, such as the medical field, forensic science, figuring out kinship relations and biomedical research (Naveed et al., 2015). The wide range of applications makes genomic data quite extraordinary in that aspect as well, and it is likely that as the knowledge increases, there will be even more use cases for genomes. It is also pretty unlikely that all genomes will be declared as public knowledge for everyone to access to, so securing them will be priority in the future as well.

Amongst other solutions for storing GD in secure way over a long period of time, different encryption methods have been in the table as well (Humbert et al., 2013). However, one problem with encryption has been the fact that *".. even the best of the cryptographic algorithms we use today could be broken in around 30 years"* (Humbert et al., 2013). For example, MD5 hash function was presented in the early 90s and first collision was found in 2004 (X. Wang, Feng, Lai, & Yu, 2004). If the same development cycle of encryption continues, any security method used today will be unsecure at some point and the system needs to be revised at some point. It would be quite useful to have a more long-lasting security method for this kind of data.

HE has been presented as an option for securing genomic data as it might make it more secure in the long run. One research paper offered an implementation of novel DTE and it was promised to hold security even at message-recovery attack (Huang et al., 2015). There was one interesting challenge to overcome at that implementation. Because the HE scheme contains a DTE, which uses very strict knowledge on the message space and the probability of each message, those features needed to be defined to genomic data as well (Humbert et al., 2013; Naveed et al., 2015). Therefore Naveed et al. (2015) made a model called genoguard, which solves the vey main problem. The solution provides a valid looking genomic data if the wrong key is used (Naveed et al., 2015) and provides a good option for long-term storing of genomes.

3.1.2 Passwords

Authentication was mentioned earlier as one of the security goals in information security. There are multiple ways to authenticate users, but the most common method today is by using passwords (Taneski et al., 2014). Taneski et al. (2014) divides passwords into textual and graphical ones. Graphical passwords may e.g contain images that the user must pick in order. This kind of authentication is called Graphical User Authentication (GUA). There is research done about images and videos used with HE (Yoon et al., 2015), so there is no reason to not do more research about how HE could be used with GUA services. The research done about HE and passwords have conserned mainly the textual passwords and for that reason the rest of this chapter will focus on textual passwords.

Insecure passwords are actually the reason why HE was developed in the first place (Juels & Ristenpart, 2014a). Passwords have multiple problems what comes to security. Some users are picking weak passwords (Taneski et al., 2014) and this makes the actual message space smaller than it is in theory. There are tools for password cracking and quite spesific information about how user are choosing their passwords (Juels & Ristenpart, 2014a). Even if the theoretical space contains all character and the lenght can be between 8-16 characters, we know that the passwords are not evenly devided among this space.

Attackers will benefit for the fact that the actual message space is smaller than the theoretical one, because that way there is less possible options to go through.This makes brute-force attack easier even if computation power is limited, as the attacker can focus on making intelligent guesses instead of going through all the possibilities. The attacker can even decide to try to use the most commonly used passwords to all of the accounts instead of focusing on cracking a spesificly This makes passwords a valid use case for HE to be used for, because passwords need more secure methods in order to guarantee their privacy in the future as well. Even though the alternative methods for passwords have been looked for some time (Bonneau et al., 2012), at the moment the passwords exists.

Password based encryption (PBE) can use passwords as part of their encryption, but it's not solely for passwords as Juels and Ristenpart (2014a) explains:*"PBE is used to protect many forms of sensitive user data ..."* (Juels & Ristenpart, 2014a). In the original paper, Juels and Ristenpart (2014a) build a scheme for PBE for RSA secret keys and credit card numbers. They go as far as to say that *"HE is particularly useful for password-based encryption (PBE)"* (Juels & Ristenpart, 2014a) and they meant any kind of data that has low min entropy keys. This makes passwords really the target area as well as one data type that could really use the extra level of security against brute-force attack.

3.1.3 Password vaults

Vaults are helping the users by making it easier to use passwords. They do this by offering one *master password* that user must remember in order to gain access to other passwords that they have stored for themselves (Golla, Beuscher, & Dürmuth, 2016; Chatterjee, Bonneau, Juels, & Ristenpart, 2015). This could potentially lead to stronger passwords in general, especially if the password vault also allows password generation (Bonneau et al., 2012). This way the password vault would use strong passwords for the other applications.

Problem with password vaults is that even if the passwords inside the vault would be strong and have high-entropy, the master password would still be chosen by the user and has the potential of being weak. This means that the master password has the same problem as the normal password has: user may pick weak passwords. As Chatterjee et al. (2015) says: *"There is no available evidence indicating that users choose significantly stronger master passwords"*. This might raise a question of: why not let the password vault generate the master password as well? This might intervene with usability issues. Users need to be able to remember the master password and the usability might become an issues with passwords like this: $MwO2FIATxidxw90UZQhmSs\&0N5D\&\&ZzFas\#joVThc*lV\#nbBkP$, because all kind of users should be able to use it easily. And besides usability, if you need to store the password somewhere else, is raises another security issue. As said by Yoon et al. (2015): *"A password that is difficult to guess is also likely to be hard to remember"*.

One example of these password vaults is LastPass. In July of 2018, while they were celebrating their 10-year anniversary, they announced that they had 16.8 million users and the user count was growing (Gott, 2018). Because password vaults contain such a sensitive data and the vaults in general have a large user base, it is important to keep them secured. After all: *"vulnerability in a password manager could allow an attacker to steal all passwords for a user in a single swoop"* (Zhiwei Li, He, Akhawe, & Song, 2014).

Password vaults have not been totally secured in the past either. According to LastPass website they have had been hacked once in 2015 (LastPass, 2019). Even though this is not scientific source, it offers valuable proof that we are not providing full security for the password vaults. At the time LastPass used PBKDF2 algorithm alongside with AES encryption (LastPass, 2019). PBKDF is especially designed to slow down attackers and make brute-force attack slower. With HE it could be possible not to only make brute-force attack slower, but prevent it all together.

Juels and Ristenpart mentioned Honey Encryption in relation to password vaults in the original article (Juels & Ristenpart, 2014). HE has been also mentioned by Harsh and Blocki as a *"Other defenses against offline attacks"* (Harsh & Blocki, 2018). If password vaults suffers from a data breach, it is vulnerable to offline brute-force attack as well (Chatterjee et al., 2015). This is because conventional encryption method will let the attacker know whether they had the right decryption key and they can keep trying until resources are depleted (Chatterjee et al., 2015).

The security issues have been raised by Chatterjee et al. (2015): *"Can password vaults be encrypted under low-entropy passwords, yet resist offline attacks?"*. Chatterjee et al. (2015) calls these kinds of vaults cracking-resistant vaults. Chatterjee et al. (2015) build a full on passwords vault scheme for cracking-resistant vault called noCrack. It was built upon the idea of HE scheme (Chatterjee et al., 2015) and quite conveniently mentions that the underlying architecture was mimicking Lastpass.

Creating an actual decoy password vault would create quite a dilemma with the other services that are meant to use password from the password vault. They would also have to be accessed with those fake vault passwords, otherwise it would be clear whether they were fake or not. After all they idea of HE is to prevent attacker for knowing when the attack was successful. Another solution is also mentioned. Instead of trying to provide actually usable password in the fake vault, let's provide a listener that will sound the alarm if any fake passwords is used (Harsha & Blocki, 2018). However, Golla et al. (2016) are presenting alarming data about the security of these so-called cracking-resistant password vaults.

In the end, password vaults could be a viable alternative to conventional passwords, although some security related factor are different. In the paper conducted by Bonneau et al. (2012) in 2012, password vault managed to get better results both in security as well as usability factors when compared against passwords(Bonneau et al., 2012). Only category that was worse was deployability.

3.1.4 Multi-dimensional data

Yoon et al. (2015) refers data such as images, videos and natural language as multi-dimensional data and that same naming convention is applied here (Yoon et al., 2015). They provide a method to use HE to encrypt multi-dimensional data and call this method *Visual honey encryption* (VHE), although in their implementation example they mainly focus on 2D images. They are keen to provide a secure method also for multi-dimensional data and in order to do this they extend the original DTE (Yoon et al., 2015). As mentioned in the paper, this new DTE could be used with images and videos.

Natural language was said to be one type of multi-dimensional data and it is used in multiple applications that may contain sensitive information. Emails, instant messages, files all contain natural language. This thesis is written in natural language. Even though this thesis is public, but there is many instances where individuals, companies and govenrments needs to be able to keep their data private. Sometimes these intervene as companies are in the hold of individuals data. It was especially said to be a challenging use case for the HE scheme(Juels & Ristenpart, 2014b). The problem of producing valid fake messages is discussed more in Chapter 3.2.1.

Instant messaging has also researched by Kim and Yoon (2016) in regards of HE scheme and Honey chatting term was invented for the systems utilizing HE. This way HE could be used with providing valid-looking natural language messages for instant messaging services. Instant messaging services such as whatsapp uses encryption in their messages. It could be useful to provide a HE scheme for this kind of service. This kind of service allows not only textual messages, but images, emojis, videos, recordings and many types of files.

3.2 Challenges of Honey Encryption

First research question in this thesis was to gather and understand the challenges that the HE is currently facing. This chapter will describe and discuss about those challenges. Solutions will work as a part to the second research question, where feasible ideas towards the implementation were needed.

Juels and Ristenpart (2014b) mention that one of the challenges in the HE implementation is the usage of Distribution-Transforming Encoder (DTE). (Juels & Ristenpart, 2014). This challenge was also mentioned in other articles: *"Building an efficient and precise DTE is the main challenge when extending HE to a real-use case"* (Naveed et al., 2015). When implementing DTE, there needs to be a message space and probability of each message defined. This might be harder than it first might seem. This is generally the main issue with HE and it will come up every time new use cases are considered to be used with HE or otherwise new implementations are made.

The original article (Juels & Ristenpart, 2014b) included a few mentions about possible challenges that HE is facing. The main challenge was the creation of valid honey messages. This was no surprise, since that same challenge existed in the Kamouflage system before (Bojinov et al., 2010) and it has been studied my multiple other papers as well, in a different context. Creation of fake data, that is virtually indistinguishable from real protected data, is not a minor task. There was also the question of how to recognize a user made typos from an attack scenario? The is a risk that typo will create a fake message which will no doubt confuse the normal users. Both of these scenarios are discussed in detail.

3.2.1 Production of honey messages

One challenge of HE is the creation of the fake messages (Juels & Ristenpart, 2014b). Juels and Ristenpart (2014b) explains that honey message is a *".. plausible-looking but bogus plaintext.."* (Juels & Ristenpart, 2014b). Here on out those fake messages are called honeywords, as those kinds of fake messages have a been called that before (Juels & Rivest, 2013), even though Juels and Ristenpart (2014b) preferred to use the term *honey message* instead. Before HE the honeywords was presented to be used as decoys, existing alongside with the correct message and therefore the attacker could not tell which one would be the correct message (Juels & Rivest, 2013). Even though HE involves distribution encoder that maps different messages (Juels, 2014) instead of storing a set of fake messages for each user (Juels & Rivest, 2013), the idea of honeyword is still quite similar.

Generally speaking the honeyword creation problem lies in the fact that the honeywords should have indistinguishability, meaning that they should be different from each other but similar in a way that the correct one cannot be picked from a group of honey messages (Juels, 2014). For example, if the correct message is a Social Security Number (SSN), the other honeywords must be some other SSNs as well. In other words, the honeywords need to be both contextually, semantically, and syntactically similar to the sugarword. The SSN must be a valid one, or the attacker can spot it as fake. The context must be carefully considered as well. If the SSN belongs to a worker in some business, there cannot be SSN that belongs to toddler or a deceased person from 1800s.

SSNs are not the problem however. It is easier to produce honeywords for data that has low-entropy and a clear structure (Juels & Ristenpart, 2014b), such as SSN, than for data that has high entropy or that lacks any structure (e.g., real-world use of natural language such as emails). Data that consist of human produced messages, such as emails are particularly problematic and this was pointed out by Juels (2014) when HE was presented: *"estimation* *of message distributions via DTEs is interesting as a natural language processing problem".* Natural language was left as unanswered problem for future research and it is still that today, as very recent study pointed out that there is not solid solutions for production of honeywords, when the data consist of human produced messages (Omolara, Jantan, & Abiodun, 2019). Human produced messages has large entropy or lacking the structure altogether.

There have been some solution ideas, however, they seem to be vulnerable to Chosen-Cipertext Attack (CCA) (Abiodun & Jantan, 2019). In CCA, the attacker is able to collect information pieces from the plaintext. (Abiodun & Jantan, 2019). This goes against the Semantic security definition, which means that with bounded resources, the attacker is unable to get any new piece of information when accessed to the corresponding ciphertext(Goldwasser & Micali, 1984). HE can achieve semantic security with low-entropy messages (Juels, 2014).

Even though the problem was mentioned in the presentation of HE as well, similar issues have existed before HE with honeywords (Juels & Rivest, 2013). Solution ideas, such as Chaffing by tweaking, Chaffing-with-a-passwords model, Hybrid generation methods and lastly the random pick (Juels & Rivest, 2013) were discussed. Random pick would fix many issues relating to passwords being too weak and the honeywords would be perfectly flat (Juels & Rivest, 2013) but this is not exactly suitable for HE as it is changing the message itself. E.g if user picks password "password123" the random pick would not actually use that as an password, but return a modificated version of that to the user and store bunch of honeywords alongside with the new password. The actual password that the user chose originally is discarded. Obviously this could work if the data can be changed during data lifetime, but it does not work e.g. with genomic data, where the actual data needs to be stored as is.

New data types require careful consideration and planning, as it is important to make sure that the DTE process is working securely with that new data. This was pointed out by Omolara et al. (2019): *"HE is tailored to work in low-entropy settings like passwords, RSA keys, PINS and Credit card numbers. Extending it to support other settings and file type requires an extensive design of the DTE [...]".*

3.2.2 User typos

Another quite critical challenge, also mentioned by Juels and Ristenpart (2014b) is the users misspelling, as the user might make typos. Typos are also a problem with Kamouflage scheme, where it was acknowledged that a typo in master password in a password vault can cause an alarm in the system and possibly some actions based on the settings, e.g the user can be locked-out (Bojino at al, 2010). If the system uses fake password fault, the typos in the password could cause a situation, where the user enters a valid looking password vault, but not his own. This is obviously quite confusing situation for the user and should be handled correctly, because typos seems to be actually quite common. As one research showed, 42% of users make at least one typos when he has to input passwords 16-22 times. (Chatterjee, Athayle, Akhawe, Juels, & Ristenpart, 2016). These situation can not be handled in the same way as the attack scenarios, as the users will get puzzled by misinformation given by the DTE.

The fact that HE produces or returns fake results if the key is incorrect may cause problems with normal users, when they accidentally misspell the password. For example, if HE is used with password vaults with a PBE encryption, and the user would mistype his password, the system would yield a fake vault for the user, which would not be the expected result and would most likely confuse the users. Therefore there are two different possible outcomes of this scenario. If it is in fact the user, then we should no doubt offer the possibility to rewrite the password. However, if it is the attacker, we should show him the fake data / send an alarm. The problem is, how do we know which situation it is? This problem was introduced in the original paper by Juels and Ristenpart (2014b) and they also presented a couple possible solutions for it, but first let's examine the problem thoroughly.

Figure 2 presents this typo problem with the HE components honeychecker and honeyword -service. It is modeled after the information presented in the original paper about HE (Juels & Ristenpart, 2014b) and the paper about honeywords (Juels & Rivest, 2013). Honeyword service is merely a component that dictates whether the input is one of the honeywords. Honeychecker contains the index of the correct honey message. If the HE system is not using honeychecker, but stores the message spaces in an inverse table, then the honey checker is not needed. The key problem with user typos is how to make a distinction between attacker just guessing the password and a misspelling? If we can be sure the input is not a honeyword, we know that the database has not been compromised. We do not know however if someone is performing a guessing/brute-force attack. Based on the definition of HE, wrong input should create a fake result.

One idea that Juels and Ristenpart (2014b) talked about as a solution for user typos was using *"online verification of plaintexts"*. This could work on items that have clear idea about the data structures, such as credit card numbers, SSN, possibly emails and so on. The idea is that those misspelled words would not exist in the database as honeywords, because those honeywords would act just like the real data. So all the honeywords would be valid credit card numbers as well. If there would be a misspelled credit card number, we can assume that it would not be an attack situation.

There is a problematic situation with online verifications. Using just the online verification would cause problems. In a scenario where a user misspells in a way that it creates a valid credit card, but not the one he has, online verification would not flag it as an misspelling and let it go forward. Honeychecker would however flag it as false and send an alarm. This is why the honeywords are needed. In order to know whether the database was breached, the password needs to be checked against the honeywords. If the misspelled item is not inside honeyword list, it is likely to be a misspelling and no alarm is needed.

Since verification service works with data that has clear structure, is is not useful solution for user typos when data consist of natural language passwords, as they are usually only constricted with length and certain ASCII characters. This make online verification service unuseful for PBE techniques overall and even with data that has a clear structure is not providing a whole solution. Other methods should definitely be looked for.

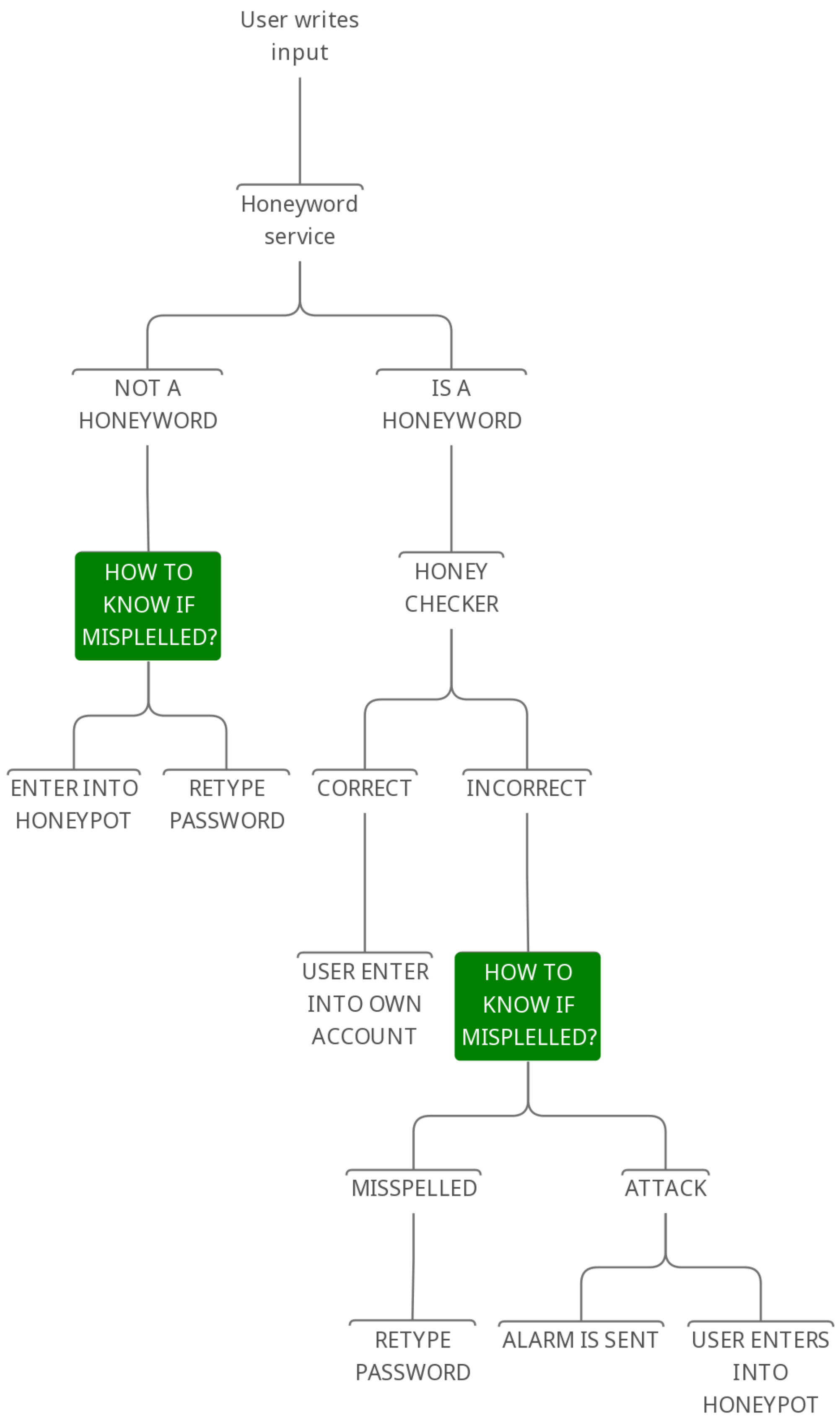

Figure 2: HE modeled with the user typo challenge

4 Honey Encryption framework

Previous chapter explained the idea of HE and explored the uses cases as well as the challenges of HE. This chapter examines the underlying framework and answers to the research questions by discussing the challenges that are related to more practical part of HE: algorithmic and deployment issues. The standalone version of HE is explained, without any other component, connections or systems taken into consideration. This enables to focus on HE in more detail and separate it from the surrounding elements, even though it is acknowledged that HE will not be this kind of independent system in real-life applications.

Theoretical framework was offered for HE implementations by Juels and Ristenpart (2014b). This framework contained a new algorithm for the encryption process called DTE-then-encrypt (Juels & Ristenpart, 2014b). This chapter explains that algorithm and it should give an understanding on why Juels and Ristenpart (2014b) considered building a secure DTE as *"non-trivial"*.

4.1 DTE-then encrypt

Before the HE framework is explain in detail, the overall idea is discussed. This will give an overview of the algorithm and give an understanding of the difference between HE and other encryption methods. After this section it should be clear why HE is merely an addition to conventional encryption methods, instead of replacement.

The basic idea of HE is to encrypt the message. Commonly noted in scientific papers, the message is referred as P (plaintext) or M (message). Ciphertext, commonly noted as C (ciphertext), is the encrypted version of plaintext. The first process of HE is to produce a seed S when used with the encryption function E. Decrypting (D) the seed in the other hand will return a message. (Juels & Ristenpart, 2014b). This makes encryption and decryption processes counter each other.

$$E(M) \rightarrow S \tag{13}$$

$$D(S) \rightarrow M \tag{14}$$

After the seed has been created it is yet again encrypted with a conventional encryption method E and a encryption key K, providing us with the final ciphertext C. (Juels & Ristenpart, 2014b). The encryption method is different than the one used earlier, and therefore it is marked as E'.

$$E'(S,K) \rightarrow C \tag{15}$$

When using the decryption algorithm, the function would work on reverse (Juels & Ristenpart, 2014b).

$$D'(C,K) \rightarrow S \tag{16}$$

This gives as the following encryption and decryption functions:

$$E(E'(M),K) \rightarrow C \tag{17}$$

$$D(D'(C,K)) \rightarrow M \tag{18}$$

The crude simplification above modeled after basic process for the HE scheme called DTE-then-encrypt (Juels & Ristenpart, 2014b). Juels and Ristenpart (2014b) added DTE as a key part of Honey Encryption. DTE is responsible for encoding and decoding the messages, and producing the seed. This makes DTE the addition to the conventional encryption methods (Juels & Ristenpart, 2014b) As the name DTE-then-encrypt implies, the idea is to use two different types of encoders: DTE and a conventional cryptographic encoder (Juels & Ristenpart, 2014b) which is presented here as a simple encryption and decryption pair. This way the overall structure of HE will be a pair on encryption and decryption functions, named here just like in the original paper: HEnc and HDec (Juels & Ristenpart, 2014b)

$$HE[DTE,(enc,dec)] = (HEnc,HDec) \tag{19}$$

```
$HEnc(K, M)$
1. $S \leftarrow \$\ encode(M)$
2. $R \leftarrow \{0,1\}^n$
3. $C_2 \leftarrow H(R,K) \oplus S$
4. $return(R, C)$
```

(a) HEnc scheme

- K Encryption key.
- M Message, plaintext.
- R Random byte array, also known as salt.
- C_2 Ciphertext.
- H Hash function.
- S Seed.
- $\oplus$ XOR
- $\leftarrow \$$ random variable is part of the process

(b) Explanations

Figure 3: HEnc framework

Juels and Ristenpart (2014b) provided algorithms for the encryption (see Figure 3) and decryption (see Algorithm 8) processes.

Algorithm in 3 is s copy of the DTE-then-encrypt framework presented by Juels and Ristenpart (2014b). It takes the encryption key K and the message M as inputs. In line number 1, the seed S is produced with the encryption method. The seed is produced from the message that was given as parameter for the function. Next, the salt is created in line 2. Salt is basically a random byte array. After the salt has been created, the salt is hashed with the encryption key K with a conventional hash algorithm. The XOR is then applied to the result of the hash and the seed, which will result the ciphertext (Juels & Ristenpart, 2014b).

Next, the algorithm is divided into parts and each part is explained with implementation examples. The examples are done with .Net framework, using C# programming language, but there is no reason to believe that there would not be equally good or inferior solution implemented with other frameworks. The idea is to provide a example and step-by-step directions so that the basic idea of the algorithm comes clear and the implementation could be possible with other methods as well.

There is also some basic concept used later on, and those concepts are defined here.

- **Message space** {M} Every possible message (Mok et al., 2017) that have >0 probability of existing (Juels & Ristenpart, 2014b). For example, if the data consists of SSN numbers in a company, the message space is all SNN numbers for adults. There would not be any invalid SSNs, so those are discarded.

- **Key space** {K} Consist on all the possible keys. Keys are usually some fixed length, depending on the algorithm (Oppliger, 2011). In HE scheme, the key and the message space should not be connected, otherwise it may be possible to connect message to decryption key (Juels & Ristenpart, 2014b).
- **Seed space** {S} Needs to be at least the same size as {M} so that every message has at least one seed (Mok et al., 2017). If a specific message has multiple seeds, the chance/probability of returning it would be greater. Therefore the seed space enables control the probability of each message getting returned (Mok et al., 2017). This is very useful for passwords, where it is known that certain passwords are picked more than others. With SSNs, the correlation should be one-to-one, since every user has only one SSN. In HE scheme, the seed space is $S = [0,1)$. (Juels & Ristenpart, 2014b) This means that seed space is divided among the messages. Then every message has a seed space between value $x, y \in [0,1)$. So in practice, messages do not have just one seed presented as a natural number, but a decimal value space.

4.2 Creation of the encryption key

The encryption algorithm HEnc will need the key and the message as inputs. For simplification reasons the process of creating the key was left out of the original algorithm, but it needs to be defined here with details, in order to provide a model for the implementation.

The length of the encryption key K should be big enough, no less than 128-bit, but preferably more, 256 or 512 bits. As an example of 256-bit creation, in figure 4 one is produced with C# and the AesManaged library that uses System.Security.Cryptography Namespace, which is built into .Net framework (MSDN2, 2006).

The result will be stored to a variable called hexKey, as a After running the code in figure 4, the variable called hexKey will contain the key in hexadecimal form, byte arrays, containing 32 bytes. Every byte is represented with 2 characters. The result will be different every time it is run, but a possible result could look like this:

$$23F61EEDC4AB09AA878385884D911EAFE30886D2972A9E40154B04C5501BC264$$

.

```
AesManaged aes = new AesManaged();
aes.GenerateKey();
var hexKey = "";
foreach (byte aesbyte in aes.Key)
        hexKey += aesbyte.ToString("X2");
```

Figure 4: Key generation with AesManaged library from .Net framework

.NET Framework has Rfc2898DeriveBytes-class that can be used for deriving keys with the PBKDF2 algorithm (MSDN2, 2006). .NET Framework 4.7.2 had an update which allows to specify the hashing algorithm used (Krishna, 2018), whereas before the user had to use only SHA-1 algorithm. Below is an example of that class, using SHA-2 as as the hash algorithm. The key can be any length byte array, but the salt has to have at least 8 bytes, resulting a 64-bit salt that can be derived from a string that has 8 characters.

```
Rfc2898DeriveBytes keyderivation = new Rfc2898DeriveBytes(key,
    salt, iterationCount, HashAlgorithmName.SHA256);
var result = keyderivation.GetBytes(32);
```

Figure 5: PBKDF2 Key derivation function example

As an example there are two results that have been gained with the function presented in figure 5. The key is simply "cat" in both times, and the salt is "saltsalt". Only difference between the two is the iteration count. In the first one the iteration count is 10000 and the one after that has 9999. As you can see, the result are almost completely different.

$$244998717B88DEC0B24F68180FFA38D715476B91973A1E6ED58C02ABBED46923$$

$$AFAC0A9A5F1687380AD06494D2A8456510545F5306EBBC507AB28C070332C8EB$$

4.3 Inverse table

Juels and Ristenpart (2014b) explains that DTE maps each message to a seed. To be more precise, each message needs to be mapped to a seed (Juels & Ristenpart, 2014b). For this to work, the message space needs to be defined. As an simple example, figure 6 presents message space containing 3 messages. This kind of example is no where representing a real message space. Better example has been made by Tyagi, Wang, Wen, and Zuo (2015), as they made an file called: *inverse_table.txt* that contained inverse table of credit card numbers. That message space contains 150 887 individual credit card numbers (Zuo, 2015) and that data looked like this:

```
38410000000004
38414000000000
374801000000007
377687000000002
372395000000001
372550000000008
[...]
8999990000000007
```

Next we need to use Cumulative Distribution Function (CDF) in order to define each message a probability between 0 and 1. CDF is bounded, like probability is in general to bounds 0 and 1. Juels and Ristenpart (2014b) defines that the first message in the message space M should have the probability of 0. Tyagi et al. (2015) do this by adding another element next to the message:

```
[(0.0,"38416000000008"),
```

Now, how do we actually calculate the seed? We use inverse sampling (Juels & Ristenpart, 2014b). Juels and Ristenpart (2014b) explains that $CDF(M_{i-1}) \leq S < CDF(M_i)$. We need to make a function that will return the seed by using CDF.

After each message has been mapped in the inverse table, it starts to look this this:

```
[..]
(0.000132549523816,"374660000000004"),
(0.000139177000007,"374661000000003"),
(0.000145804476197,"370266000000006"),
[..]
```

The seed space must also be defined. Juels and Ristenpart (2014b) said that the seed could be binary strings, a set of 0's and 1's. The seed space must be at least at large as the message space, so that each message can be mapped to individual seed. Some messages will have multiple seeds, and each seed must be mapped to a message. This way, when decrypting, the seed will always return a value.

4.4 Distribution-transforming encoder (DTE)

Juels and Ristenpart (2014b) explained that DTE has a similarity to arithmetic and Huffman coding (Juels & Ristenpart, 2014b). Huffman coding compacts larger data sets by using frequency distribution and replacing each item with a code. The codes are stored in a file, so the recovery for the original message can be done (Robling Denning, 1982, p.18; Cormen, Leiserson, Rivest, & Stein, 2009, p. 428).

Development of secure DTE will be the key challenge when implementing HE as Juels and Ristenpart (2014b) warned: *"Building DTEs is non-trivial in many cases"*. Juels and Ristenpart (2014b) made clear that decryption process needs to be deterministic and therefore there will not be randomness involved. Instead it is crucial that if a message is encoded and then decoded, it will always return the same message.

$$Pr[decode(encode(M)) = M] = 1, M \in \{M\} \tag{20}$$

As mentioned above, DTE uses a method in order to transform seed values to messages or messages to seed values (Juels & Ristenpart, 2014b) depending on is it encrypting or decrypting. If the system uses one-to-one correlation between seeds and messages, basically each seed is equivalent to a single message and vice versa. If the seed space is larger than the message space, then some messages have multiple corresponding seeds, in that way that all messages have at least one.

DTE uses inverse sampling (Juels & Ristenpart, 2014b), which means mapping values from another set into another set. Here the sets are the message space M and the seed space S. Figure 6 shows this mapping with a message space containing 3 messages. Similar figures have been presented by (Juels & Ristenpart, 2014b; Mok et al., 2017), with only minor differences concerning the data or number if seeds.

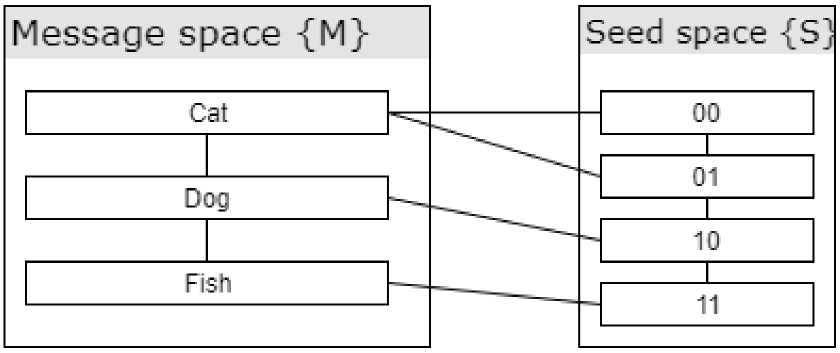

Figure 6: Inverse sampling

Important part of the mapping is that the message space contains all the messages that have some probability of existing (Juels & Ristenpart, 2014b) and that each message has at least one seed mapped to it (Juels & Ristenpart, 2014b).

4.5 Hashing

Use conventional hash function to ensure symmetric encryption. Symmetric cipher can be constructed with hash functions (Katz et al., 1996, p. 31). Hash function uses a random salt and a key.

Finally hash-function is used for the combination of salt and the key. (Juels & Ristenpart, 2014b). Juels and Ristenpart (2014b) recommend using Key Derivation Function (KDF) for the key before hashing it, but they did leave it out of the equation for simplification reasons. There is some restriction for the conventional encryption, explained by Juels and Ristenpart (2014b):*"This conventional encryption scheme enc must have message space equal to the seed space and all ciphertexts must decrypt under any key to a valid seed."*

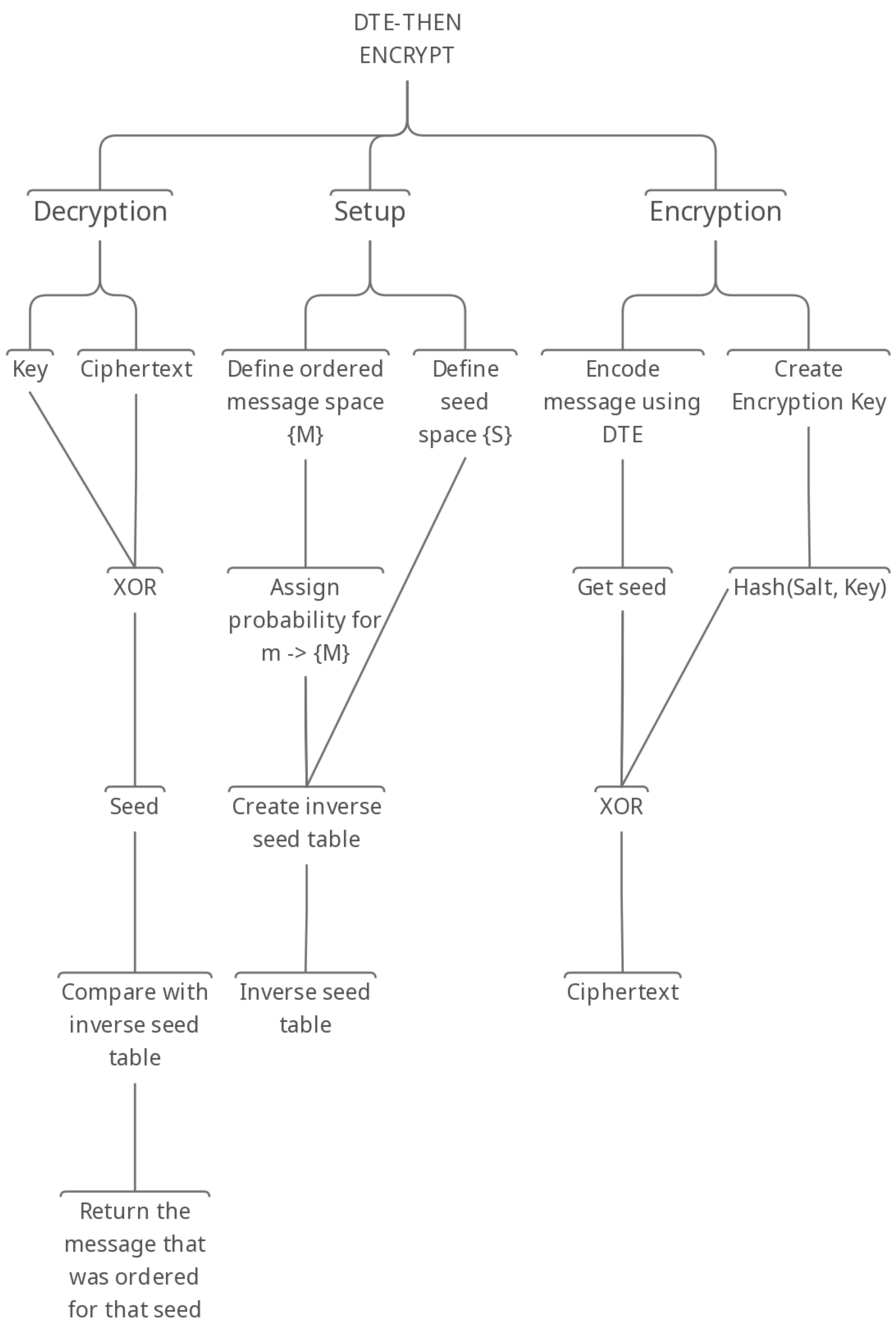

Figure 7: DTE-then encrypt framework

4.6 Decryption

```
$HDec^H(K,(R,C_2))$
$S \leftarrow C_2 \oplus H(R,K)$
$M \leftarrow decode(S)$
$return\ M$
```

(a) HDec scheme

- K Encryption key.
- M Message, plaintext.
- R Random byte array, also known as salt.
- C_2 Ciphertext.
- H Hash function.
- S Seed.
- $\oplus$ XOR
- $\leftarrow \$$ random variable is part of the process

(b) Explanations

Figure 8: HEnc framework

Decryption part of HE must also be discussed as it is quite different from the encryption function HEnc. The algorithm for the decryption process is presented in figure 8, which is a copy of the original HDec algorithm, presented by Juels and Ristenpart (2014b). Juels and Ristenpart (2014b) designed that the decryption process will follow a conventional decryption process, if the key is correct. However, if the key is incorrect, the HDec process should return a sweetword instead. As an example, if we use the simple examples

Overall the decryption has an important role of returning the sweetwords in a secure and reasonable way (Juels & Ristenpart, 2014b). It should also be noted that while the encryption process uses quite a lot of randomness in multiple places, the decryption process should always be deterministic without any randomness.

Previously we defined an inverse table. That table is needed in the decryption process, because CDF work on one-way only. Given a message it can return a probability, like we need to do in the encryption process. In the decryption phase we have a hold of the probability, the seed. CDF can not return a message with the probability. This is why we made the inverse sampling table. Tyagi et al. (2015) explains that with inverse sampling table, a message can be searched. They also mentioned that it is possible to use binary search to the seed values first, in order to make the search more efficient (Tyagi et al., 2015).

5 Conclusions and future work

This thesis looked into the challenges and solutions of Honey Encryption by going through the preceding research. In 2014, the Honey encryption was said to provide protection against brute-force attacks, because the attacker is unable to know when he has decrypted the data with a valid key. Today that statement still stands, as there was no evidence in contradicting the security statements done in the original paper by Juels and Ristenpart (2014b). HE scheme will be beneficial to possibly multiple data types and use cases, as the original paper already provided theoretical implementations for passwords, RSA keys and PINs. Since the original paper, it has been used also with genomic data, pictures, videos and with natural language. New application types has been applied with IoT-devices and public cloud storages. The popularity of new use cases for HE offers an insight of the interest and the potential of the HE sheme.

Besides the potential, there is a actual need for an additional security in all of these use cases, as it is necessary to provide privacy no matter what kind of resources the attacker has. As we are moving more and more into technology based world and there is data that should not ever be leaked out, the barrier for brute-force attack becomes critical. Besides the one-time pad, there hasn't been a usable solution for securing data against brute-force attack, and even the one-time pad is arguably unusable because of the storage space it needs. Kamouflage is lacking the mathematical proof and the decoy honeywords has some unanswered issues on what is the best method for producing the honeywords.

Even though HE has great potential, it can be quite difficult to implement it in practice, because there is no precise implementation directions. How to actually build a secure DTE? How to calculate the probability of the messages? How to make sure that the attacker will not know fake message from a real one? These are good questions, because if the HE is implemented in a insecure way, the security will be go down to the level of of the conventional encryption. Even though the security of the conventional encryption should always be there, the resources used for the implementation of HE will be wasted. Poorly implemented HE framework can also provide false sense of security and be harmful in overall for the application, if it is believed to be brute-force proofed and it is not. There is no simple test case for making sure the brute-force attack has been prevented.

Implementation of HE in a way that makes the system more secure needs designing and possibly some extra resources. As mentioned by Harsha and Blocki: *"These methods require the purchase of additional equipment, which may prevent those with more limited financial resources from employing them"* (Harsha & Blocki, 2018). There are currently no usable and ready-to-use tools that would provide easy start for the HE implementation. This means that the service would have to be build from scratch. These factors will no doubt limit the possible users of HE to those that are willing to spend the necessary resources. If the large-scale usage of HE is ever desired, there has to be easy-to deploy libraries or other services that would enable developers to quickly implement HE in the system.

Overall this thesis contained information about the current state of the HE scheme. There are still unresolved challenges relating to HE, mostly with user typos and the use of natural language encoder. These were presented in this thesis, but there were no absolute solutions for these challenges. More research should definitely be done so that these issues can be addressed. At this state HE has shown potential to be a secure addition to conventional encryption and it has a great diversity of possible use cases and it has attracted some researchers. This undoubtedly means that the research of HE will go on, as it should be. There is a need for research done with different DTE frameworks for different data types. The work for honey encryption is not yet over, nor it will over for data security for a long time.

Besides HE, there is a need for general research on increasing password security or how to move away from conventional passwords. In order for the Kamouflage system to be a viable option, the mathematical proof should be provided and the security issues mentioned in (Golla et al., 2016) must be addressed. Besides the HE, the quantum encryption (Gisin et al., 2002) could provide the real security solutions in the future.

Bibliography

Abiodun, E. O., & Jantan, A. (2019). Modified honey encryption scheme for encoding natural language message. *International Journal of Electrical and Computer Engineering*, *9*(3), 1871.

Almeshekah, M. H., & Spafford, E. H. (2014). Planning and integrating deception into computer security defenses. In *Proceedings of the 2014 New Security Paradigms Workshop* (pp. 127–138). ACM. Retrieved from https://dl.acm.org/citation.cfm?id=2683482

Aumasson, J.-P. (2017). *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press.

Barros, A. (2003). RES: Protocol Anomaly Detection IDS—Honeypots.

Bidgoli, H. (2006). *Handbook of Information Security, Information Warfare, Social, Legal, and International Issues and Security Foundations*. John Wiley & Sons.

Bojinov, H., Bursztein, E., Boyen, X., & Boneh, D. (2010). Kamouflage: Loss-resistant password management. In *European symposium on research in computer security* (pp. 286–302). Springer.

Bonneau, J., Herley, C., Van Oorschot, P. C., & Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy* (pp. 553–567). IEEE. Retrieved from https://ieeexplore.ieee.org/document/6234436

Chatterjee, R., Athayle, A., Akhawe, D., Juels, A., & Ristenpart, T. (2016). pASSWORD tYPOS and how to correct them securely. In *2016 IEEE Symposium on Security and Privacy (SP)* (pp. 799–818). IEEE. Retrieved from https://ieeexplore.ieee.org/document/7546536

Chatterjee, R., Bonneau, J., Juels, A., & Ristenpart, T. (2015). Cracking-resistant password vaults using natural language encoders. In *2015 IEEE Symposium on Security and Privacy* (pp. 481–498). IEEE. Retrieved from https://ieeexplore.ieee.org/document/7163043

Cheswick, B. (1992). An Evening with Berferd in which a cracker is Lured, Endured, and Studied. In *Proc. Winter USENIX Conference, San Francisco* (pp. 20–24).

Choi, H., Nam, H., & Hur, J. (2017). Password typos resilience in honey encryption. In *2017 International Conference on Information Networking (ICOIN)* (pp. 593–598). IEEE. Retrieved from https://ieeexplore.ieee.org/document/7899565

Cohen, F. (2006). The use of deception techniques: Honeypots and decoys. *Handbook of Information Security*, *3*(1), 646–655.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press.

Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, *22*(6), 644–654. Retrieved from https://ieeexplore.ieee.org/document/1055638

Dougherty, C. R. (2009). Vulnerability Note VU# 836068 MD5 vulnerable to collision attacks. *Retrieved August*, *26*, 2009.

Finnish Advisory Board on Research Integrity. (2012). Responsible conduct of research and procedures for handling allegations of misconduct in Finland. Finnish Advisory Board on Research Integrity Helsinki.

Florencio, D., & Herley, C. (2007). A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web* (pp. 657–666). ACM. Retrieved from https://dl.acm.org/citation.cfm?id=1242661

Gisin, N., Ribordy, G., Tittel, W., & Zbinden, H. (2002). Quantum cryptography. *Reviews of modern physics*, *74*(1), 145.

Goldwasser, S., & Micali, S. (1984). Probabilistic encryption. *Journal of computer and system sciences*, *28*(2), 270–299.

Golla, M., Beuscher, B., & Dürmuth, M. (2016). On the security of cracking-resistant password vaults. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1230–1241). ACM. Retrieved from https://dl.acm.org/citation.cfm?id=2978416

Gott, A. (2018). LastPass celebration. Retrieved from https://blog.lastpass.com/2018/07/celebrating-10-years-lastpass.html/

Han, X., Kheir, N., & Balzarotti, D. (2018). Deception Techniques in Computer Security: A Research Perspective. *ACM Computing Surveys (CSUR)*, *51*(4), 80. Retrieved from https://dl.acm.org/citation.cfm?id=3214305

Harsha, B., & Blocki, J. (2018). Just In Time Hashing. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)* (pp. 368–383). IEEE. Retrieved from https://ieeexplore.ieee.org/document/8406611

Huang, Z., Ayday, E., Fellay, J., Hubaux, J.-P., & Juels, A. (2015). GenoGuard: Protecting genomic data against brute-force attacks. In *2015 IEEE Symposium on Security and Privacy* (pp. 447–462). IEEE. Retrieved from https://ieeexplore.ieee.org/document/7163041

Humbert, M., Ayday, E., Hubaux, J.-P., & Telenti, A. (2013). Addressing the concerns of the lacks family: quantification of kin genomic privacy. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 1141–1152). ACM. Retrieved from https://dl.acm.org/citation.cfm?id=2516707

Isomäki, H. (2018). Tutkimussuunnitelmien-yleiskatsaus-harjoitustyon-ohjeistusta. Lecture notes, TIES501 Seminar, University of Jyväskylä.

Jo, H.-J., & Yoon, J. W. (2014). Poster: statistical coding scheme for the protection of cryptographic systems against brute-force attack. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*. Retrieved from https://ieeexplore.ieee.org/document/6859779

Jo, H.-J., & Yoon, J. W. (2015). A new countermeasure against brute-force attacks that use high performance computers for big data analysis. *International Journal of Distributed Sensor Networks*, *11*(6), 406915.

Juels, A. (2014). A bodyguard of lies: the use of honey objects in information security. In *Proceedings of the 19th ACM symposium on Access control models and technologies* (pp. 1–4). ACM. Retrieved from https://dl.acm.org/citation.cfm?id=2613088

Juels, A., & Ristenpart, T. (2014a). Honey encryption: encryption beyond the brute-force barrier. *IEEE Security & Privacy*, *12*(4), 59–62. Retrieved from https://ieeexplore.ieee.org/document/6876246

Juels, A., & Ristenpart, T. (2014b). Honey encryption: Security beyond the brute-force bound, 293–310.

Juels, A., & Rivest, R. L. (2013). Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 145–160). ACM.

Kaliski, B. (2000). Password-based cryptography specification. *RFC 2898*.

Katz, J., Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC press. Retrieved from https://doi.org/10.1201/9781439821916

Kim, J.-I., & Yoon, J. W. (2016). Honey chatting: A novel instant messaging system robust to eavesdropping over communication. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2184–2188). IEEE. Retrieved from https://ieeexplore.ieee.org/document/7472064

Kolata, G., & Murphy, H. (2018). The golden state killer is tracked through a thicket of DNA, and experts shudder. *The New York Times*, *27*.

Krishna, P. (2018). Announcing the .NET Framework 4.7.2, .Net Blog. Microsoft Developer Network 2. Retrieved from https://devblogs.microsoft.com/dotnet/announcing-the-net-framework-4-7-2/

LastPass. (2019). LastPass homepage. Retrieved from https://www.lastpass.com/enterprise/security

Li, Z. [Zhigong], Han, W., & Xu, W. (2014). A large-scale empirical analysis of chinese web passwords. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)* (pp. 559–574).

Li, Z. [Zhiwei], He, W., Akhawe, D., & Song, D. (2014). The emperor's new password manager: Security analysis of web-based password managers. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)* (pp. 465–479).

Lukka, K. (2003). The constructive research approach. *Case study research in logistics. Publications of the Turku School of Economics and Business Administration, Series B*, *1*(2003), 83–101.

Martin, K. (2012). Everyday Cryptography Fundemental Principles and Applications (2012). Oxford University Press.

Microsoft Docs. (n.d.). SHA256 Class (System.Security.Cryptography). Retrieved April 24, 2019, from https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.sha256?view=netframework-4.8

Mitnick, K. D., & Simon, W. L. (2011). *The art of deception: Controlling the human element of security*. John Wiley & Sons.

Mok, E., Samsudin, A., & Tan, S.-F. (2017). Implementing the honey encryption for securing public cloud data storage. In *In Proceedings of First International Conference on Computer Science and Engineering*.

Moriarty, K., Kaliski, B., & Rusch, A. (2017). *PKCS# 5: password-based cryptography specification version 2.1*.

MSDN2. (2006). System.security.cryptography namespace. Microsoft Developer Network 2. Retrieved from http://msdn2.microsoft.com/en-us/library/%20system.security.cryptography.aspx

Naveed, M., Ayday, E., Clayton, E. W., Fellay, J., Gunter, C. A., Hubaux, J.-P., … Wang, X. (2015). Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, *48*(1), 6. Retrieved from https://dl.acm.org/citation.cfm?id=2767007

Omolara, A. E., Jantan, A., & Abiodun, O. I. (2019). A comprehensive review of honey encryption scheme. *Indonesian Journal of Electrical Engineering and Computer Science*, *13*(2), 649–656.

Omolara, A. E., Jantan, A., Abiodun, O. I., & Poston, H. E. (2018). A novel approach for the adaptation of honey encryption to support natural language message. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (Vol. 1).

Oppliger, R. (2011). *Contemporary cryptography*. Artech House.

Robling Denning, D. E. (1982). *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc.

Schneier, B. (2011). *Secrets and lies: digital security in a networked world*. John Wiley & Sons.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, *27*(3), 379–423.

Sotirov, A., Stevens, M., Appelbaum, J., Lenstra, A. K., Molnar, D., Osvik, D. A., & de Weger, B. (2008). *MD5 considered harmful today, creating a rogue CA certificate*.

Spitzner, L. (2001). The value of honeypots, part one: Definitions and values of honeypots. *Security Focus*.

Stevens, M., Bursztein, E., Karpman, P., Albertini, A., & Markov, Y. (2017). The first collision for full SHA-1. In *Annual International Cryptology Conference* (pp. 570–596). Springer.

Stoll, C. (1989). *The cuckoo's egg: tracking a spy through the maze of computer espionage*. Simon and Schuster.

Taneski, V., Heričko, M., & Brumen, B. (2014). Password security—No change in 35 years? In *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1360–1365). IEEE. Retrieved from https://ieeexplore.ieee.org/document/6859779

Tyagi, N., Wang, J., Wen, K., & Zuo, D. (2015). Honey encryption applications. *Network Security*.

University of Jyväskylä. (2019). Privacy notice information. Retrieved from https://www.jyu.fi/en/university/privacy-notice/jyx-archive

Van Leeuwen, J., & Leeuwen, J. (1990). *Handbook of theoretical computer science*. Elsevier.

Wang, D., Cheng, H., Wang, P., Yan, J., & Huang, X. (2018). A security analysis of honeywords. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium*.

Wang, X., Feng, D., Lai, X., & Yu, H. (2004). Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. *IACR Cryptology ePrint Archive*, *2004*, 199.

VU, C. V. N. (2014). 836068. *Retrieved February*, *8*.

Yoon, J. W., Kim, H., Jo, H.-J., Lee, H., & Lee, K. (2015). Visual honey encryption: Application to steganography. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security* (pp. 65–74). ACM. Retrieved from https://dl.acm.org/citation.cfm?id=2756606

Yuill, J. J. et al. (2007). Defensive computer-security deception operations: Processes, principles and techniques.

Yuill, J., Zappe, M., Denning, D., & Feer, F. (2004). Honeyfiles: deceptive files for intrusion detection. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.* (pp. 116–122). IEEE. Retrieved from https://ieeexplore.ieee.org/document/1437806

Zuo, D. (2015). honeyencryption. *Github*. Retrieved from https://github.com/danielzuot/honeyencryption