Leveraging an Active Directory for the Generation of Honeywords

Johan Lundström

Information Security, master's level (120 credits)
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Abstract

Honeywords, fake passwords that when used by an adversary are set to trigger an alarm, is one way of detecting security breaches. For them to be effective, however, they must resemble real passwords as closely as possible and thus, the construction of the honeywords is crucial.

In this thesis, a new model for generating honeywords, PII-Syntax, is presented that was built in part on a previous model but reworked and adapted to meet new requirements. The purpose of the study was to investigate whether an Active Directory, (AD) could be used as a resource in the construction of honeywords. The assumption was that the AD contains information about real system users that could be leveraged to create high-quality honeywords because of the very fact that they are based on actual users. It is a well-known fact that many users have a natural inclination towards incorporating personal information when choosing their passwords, information that can be leveraged by an adversary making the passwords easier to retrieve. The proposed model capitalizes on this fact and bases the honeyword generation process on users’ personally identifiable information, PII. The motivation for this is to enhance the quality of the honeywords, i.e. making them more plausible from the perspective of the adversary.

The resulting model performed equally well or better than all existing honeyword generation algorithms to which it was compared with regard to flatness, DoS resistivity, multiple system vulnerability and storage cost. The most important contribution, however, is the inclusion of users’ personal information in the generation of the honeywords that ultimately help strengthen the security of password-based authentication systems.

Contributions from this thesis include a novel manner in which to approach a well-known problem, both in a theoretical as well as a practical sense: PII-Syntax is a new honeyword generation algorithm that apart from performing equally well or better than previous algorithms brings an added value of believability to the generated honeywords because of the inclusion of users’ personal information found in an AD.

Keywords: honeywords, passwords, authentication, password-based authentication, password cracking, deception techniques
Acknowledgements

First of all, a big thanks to LTU for the incredible learning experience that I have been fortunate to experience for the last four years culminating in this work.

Next, I wish to thank my supervisor professor Maung Sein for the continued support and valuable feedback during the writing process.

Finally, I wish to thank my family, Ulrika and my daughter Giulia for all your patience and support and for giving me the opportunity to pursue a new career. This accomplishment would not have been possible without you. Love always.
# Contents

Abstract ................................................................................................................................. i

Acknowledgements ............................................................................................................... ii

Abbreviations ....................................................................................................................... v

List of figures ......................................................................................................................... vi

List of tables ........................................................................................................................... vii

1. Introduction ........................................................................................................................ - 1 -
   1.1 Background .................................................................................................................. - 1 -
   1.2 Problem and purpose ................................................................................................. - 2 -
   1.3 Hypothesis .................................................................................................................. - 3 -
   1.4 Research question and delimitations ......................................................................... - 3 -
   1.5 Contributions ............................................................................................................. - 4 -
   1.6 Thesis outline ............................................................................................................. - 5 -

2. Literature Review .............................................................................................................. - 6 -
   2.1 Establishing context .................................................................................................... - 6 -
   2.2 Existing algorithms for the generation of honeywords .............................................. - 7 -
   2.3 Metrics for evaluation ............................................................................................... - 13 -
      2.3.1 Security standards ............................................................................................... - 13 -
      2.3.2 Usability standards ............................................................................................. - 15 -
      2.3.3 Storage cost ....................................................................................................... - 16 -
   2.4 Algorithm comparison and evaluation ...................................................................... - 16 -
   2.5 Passwords .................................................................................................................. - 21 -
      2.5.1 Introduction ....................................................................................................... - 21 -
      2.5.2 Characteristics in user password construction .................................................... - 22 -
      2.5.3 Personal information in passwords .................................................................... - 24 -

3. Method ............................................................................................................................... - 26 -
   3.1 Method selection ........................................................................................................ - 26 -
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Active Directory</td>
</tr>
<tr>
<td>ADR</td>
<td>Action Design Research</td>
</tr>
<tr>
<td>APT</td>
<td>Advanced persistent threat</td>
</tr>
<tr>
<td>CNF</td>
<td>Close-number-formation</td>
</tr>
<tr>
<td>DBIR</td>
<td>Data Breach Investigations Report</td>
</tr>
<tr>
<td>DDL</td>
<td>Data Definition Language</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DSRM</td>
<td>Design science research methodology</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics processing unit</td>
</tr>
<tr>
<td>HBAT</td>
<td>Honeyword based authentication technique</td>
</tr>
<tr>
<td>IAM</td>
<td>Identity &amp; Access Management</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>MSV</td>
<td>Multiple System Vulnerability</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>PCFG</td>
<td>Probabilistic Context-Free Grammars</td>
</tr>
<tr>
<td>PDP</td>
<td>Paired Distance Protocol</td>
</tr>
<tr>
<td>PII</td>
<td>Personally Identifiable Information</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Lifecycle</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hashing Algorithm</td>
</tr>
<tr>
<td>SSN</td>
<td>Social Security Number</td>
</tr>
<tr>
<td>SSO</td>
<td>Single-Sign-On</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
</tbody>
</table>
List of figures

Figure 1 – Honey Circular List (Chakraborty & Mondal, 2017, p. 159)........................ - 12 -
Figure 2 – The Software Development Life Cycle (Stephens, 2015, p. 277)........... - 29 -
Figure 3 – Testing environment setup ...................................................................... - 31 -
Figure 4 – Sweetwords relationships – sugarword ....................................................... - 35 -
Figure 5 – Sweetwords relationships – shamword....................................................... - 36 -
Figure 6 – Shamword constituents ............................................................................. - 36 -
Figure 7 – Modelling-syntax process model ............................................................... - 39 -
Figure 8 – Holistic overview of PII-Syntax............................................................... - 43 -
Figure 9 – Class diagram for PII-Syntax................................................................. - 44 -
Figure 10 – Honeywords based on the shamword alone .......................................... - 62 -
Figure 11 – Two examples of honeyword distribution ............................................. - 63 -
List of tables

Table 1 – Comparison of honeyword-generation methods, adopted from (Juels & Rivest, 2013, p. 10) ...........................................................................................................................- 17 -

Table 2 – Flatness under type A₁ and A₂ attackers, adapted from (D. Wang et al., 2017, p. 13).................................................................................................................................- 18 -

Table 3 – Comparative analysis of methods (Chakraborty & Mondal, 2016, p. 807)- 19 -

Table 4 – Quantitative comparison of methods (Chakraborty & Mondal, 2017, p. 166) .................................................................................................................................- 20 -

Table 5 – Quantitative comparison of methods (Chakraborty & Mondal, 2017, pp. 166–167).................................................................................................................................- 21 -

Table 6 – PII attributes ..............................................................................................................................................- 35 -

Table 7 – Performance evaluation of modelling-syntax, CNF and Caps-key........ - 38 -

Table 8 – Token attributes .........................................................................................................................................- 39 -

Table 9 – Process model outcome.............................................................................................................................- 40 -

Table 10 – Number of formula permutations .............................................................................................................- 41 -

Table 11 – Permissible token modification methods ....................................................................................................- 42 -

Table 12 – Token positioning .......................................................................................................................................- 45 -

Table 13 – Honeyword methods’ probability rate (in %)...............................................................................................- 49 -

Table 14 – Comparison of PII-Syntax to other honeyword algorithms......................................................... - 57 -
1. Introduction

1.1 Background

Over the last couple of years, it has become more and more common to hear about media reporting of password breaches. LinkedIn, Adobe and Yahoo to just name a few are some of the big companies who have suffered from the theft of password files and the subsequent disclosure of their users’ passwords (Gaylord, 2012; D. Wang, Cheng, Wang, Yan, & Huang, 2017). According to the annual report on data breaches issued by well-known communications enterprise Verizon, user passwords and credentials are still a high-profile target for adversaries (‘2017 DBIR’, 2017). For a number of sectors, the top target of choice was user credentials, peaking at over 70% in the financial sector (ibid.). Moreover, it is still common practice for users to use the same password for multiple sites (Bonneau, Herley, Van Oorschot, & Stajano, 2015; Shen, Yu, Xu, Yang, & Guan, 2016), meaning that a single compromised password can allow an adversary to gain access to numerous accounts, potentially disclosing sensitive and private data (Taneski, Hericko, & Brumen, 2014; Xiaoying Yu & Qi Liao, 2016). To make matters worse, it is not uncommon using the same password for private and organizational accounts, a practice that may put company data at risk (T. Campbell, 2016; Ives, Walsh, & Schneider, 2004).

It is a long established fact that system and account authentication using passwords is not a preferred method in terms of security (Bonneau et al., 2015), as users normally tend to choose weak passwords (Genç, Kardaş, & Kiraz, 2017; Juels & Rivest, 2013) that given the computational resources at an adversary’s disposal today are almost destined to be compromised. Nevertheless, password authentication is still the most common method of authentication around today and despite the known weaknesses will most likely continue to be so for the foreseeable future (Castelluccia, Chaabane, Dürmuth, & Perito, 2013; Herley & Van Oorschot, 2012). There is also the threat of targeted password guessing that has proven to be even more effective in cracking user passwords (Castelluccia et al., 2013) and which is even more difficult to defend against (Bonneau et al., 2015). Advancements in GPU technologies especially have pushed the boundaries for what is feasible to crack, necessitating stronger hashing algorithms and lengthier salts1. And since password authentication is so commonly used, it will continue to be an attractive target for adversaries seeking to gain access to secure systems and acquire valuable resources.

There is a direct correlation between the complexity of a password and the amount of effort required in breaking it. The longer and more abstract, generally the more effort is required to break it (Bonneau et al., 2015). However, since these types of passwords

---

1 Salt in the context of hashing refers to the generation of a random value that is combined with the password to make up the hash. The password and the salt are both inputted to the hashing algorithm which generates a fixed-length value. The salt is used for three reasons (Stallings & Brown, 2015): any duplicate passwords become obfuscated; conducting an offline dictionary attack becomes much more difficult; and detecting whether a user with accounts on multiple systems has reused the passwords becomes almost impossible.
make it harder for the user to remember, in the interest of usability, weaker passwords are normally chosen.

Another matter is the protection of the passwords which are typically stored in a list on a separate server. The common way in use today is not to store the passwords in cleartext but in hashes. This makes it harder for an adversary to make use of them, should they be compromised, provided a strong hashing algorithm such as SHA-2\(^2\) or its soon-to-come successor SHA-3 is used of course. Unfortunately, however, there is ample evidence to suggest that the practice of hashing the passwords and using salt is not always followed, despite the potential consequences (Bonneau et al., 2015).

The use of honeypots, servers configured to attract the attention of attackers in an effort to study their behaviour, skills, methods and intentions have been around for some time. In 2013 Juels & Rivest introduced what might be viewed as an extension of this, the concept of honeywords, fake passwords that when used by an adversary are set to trigger an alarm, alerting the administrator of a password file compromise (Juels & Rivest, 2013). They suggested several different methods for honeyword generation which since have been refined and expanded on by other researchers (Chakraborty & Mondal, 2017; Erguler, 2016; Genç et al., 2017) to name a few.

It might seem strange to focus the attention on constructing fake passwords to make the life difficult for any unauthorized person trying to make use of them, as this presupposes that the list has both been disclosed and the passwords cracked. However, as indicated above, this is unfortunately not that uncommon and as long as password-based authentication continues to be used, password breaches will continue to be a serious threat to secret organizational information and intellectual property.

As one weapon to counter attacks using stolen passwords, this study has tried to enhance the quality of the fake passwords which will have a direct impact in helping to detect password list breaches. As the intended setting is in any organization that utilizes some form of an Active Directory (AD) to manage staff information, the intention and ambition are that it will ultimately help protect sensitive and valuable company data.

1.2 Problem and purpose

In order to protect user passwords from being exploited once a password list has been disclosed and its contents cracked, another line of defence is necessary. The use of fake passwords and or accounts, i.e. honey accounts or honeywords respectively, is one way of making an attacker’s life more difficult (Juels & Rivest, 2013). For honeywords to be effective, however, they must resemble real passwords as closely as possible so that an attacker will mistakenly take them for real ones and decide to use them in an effort to gain access to a system or account. Thus, the actual construction of the honeywords is crucial.

---

\(^2\) SHA (Secure Hash Algorithm) produces hash values of different lengths; SHA-1 produces hash values of 160 bits whereas SHA-2 yields lengths from 256 up to 512 bits.
A number of methods and algorithms on the construction of honeywords exist, that each has its benefits and drawbacks. Some of them rely on the user to remember some extra information to increase the security. By using the information in an AD and automating the honeyword generation process, the user is alleviated of the duties of any extra effort which ultimately helps improving usability.

The purpose of this study has been to investigate if and how an Active Directory could be used as a resource in the construction of honeywords, whether its use would lead to an improvement over existing honeyword generation methods. The assumption was that the AD contains information about real system users, information that could be leveraged in the creation of fake passwords that will appear real because of the very fact that they are based on actual users.

### 1.3 Hypothesis

The underlying hypothesis has been that utilizing information found in an AD would be beneficial to the construction of honeywords. This would subsequently enhance the level of security, especially in cases where the attacker is in possession of some knowledge of company staff records. This information might have been acquired from an insider, through social engineering, malware or by simply surfing the company website. Or, the attacker himself is an insider.

### 1.4 Research question and delimitations

The specific research question that the thesis set out to answer was:

- Can the information found in a typical AD be leveraged to good effect in an algorithm for the construction of honeywords?

In order to answer the research question, a design science approach was employed to develop an algorithm for the generation of honeywords based on users’ personal information typically found in an AD. The algorithm was evaluated against metrics typically used in this context and also compared to previous honeyword generation algorithms.

The results from the study clearly indicate that the inclusion of an AD in the honeyword generation model does indeed provide an added value when compared to existing models. Apart from the fact that the proposed model performs equally well or better than present models when evaluated against relevant metrics, the inclusion of PII for the creation of honeywords is very much in alignment with how users tend to choose passwords.

The general assumption of the report is that “better” honeywords will increase the likelihood that an adversary will make use of them when trying to gain system access, thus leading to subsequent detection. Better in this case would mean genuine, plausible passwords that appear as “realistic” as possible and cannot be distinguished from a user’s real one. Since the advantage of using honeywords becomes apparent only after
the compromise of a password list, the following assumptions are made as to the scenario in which they can play a significant part in securing system access:

- The adversary has gotten hold of a password list
- The adversary has cracked the list if necessary, i.e. inverted the hashing algorithm with or without salt
- The adversary tries to gain access by using the compromised passwords
- The adversary has obtained some knowledge of the company, perhaps as part of some initial reconnaissance, social engineering or by the use of APTs, that he will try to leverage to his advantage
- The adversary might be an insider

There are a number of different ways to protect company and organizational data from being breached, just as there are a multitude of options on how to secure password-protected systems. The use of honeywords is not to be mistaken for the only or the best alternative, rather it should be viewed as an effective measure that works best in conjunction with other methods, providing a layered defence or defence-in-depth. The other measures are not discussed in this thesis; the focus has been solely on the use of honeywords.

For increased security of password lists, they are typically stored in a hashed format, preferably also using salt, as noted earlier. However, as the purpose of the study has been the generation of plausible honeywords, the strength of the particular hashing algorithm itself has not been considered. The actual protection of the passwords lists was neither within the scope of this thesis.

To be able to create passwords that appear real, they must resemble passwords that are actually used by real users. Studies of the topic exist and no new investigation into this area was conducted, instead the findings there were used.

The focus has neither been on how to make users choose strong passwords, or what constitutes a “strong” password, as this has been discussed at length in several other works, including (Bonneau, 2012; Bonneau et al., 2015; Shen et al., 2016), to name but a few.

1.5 Contributions

The outcome from this work is twofold. First, the personal information from an AD is leveraged in order to create realistic-looking passwords. Second, an algorithm that generates honeywords based on the passwords from the first phase is first designed conceptually before ultimately being implemented in code. The implementation made it possible to evaluate the artefact against certain established measurements and criteria. It should be noted, however, that the main focus was not on producing the best and most effective implementation possible but rather to be able to demonstrate the solution in practice, i.e. much like a proof of concept. Contributions include a novel manner in which to approach a well-known problem, both in a theoretical as well as a practical sense.
1.6 Thesis outline

After this introductory chapter, the second chapter seeks to establish a solid theoretical background through a literature review that explores the two topics of honeywords and password usage. Next, the method chapter explains the method used to carry out the project. The fourth chapter shows the results beginning with a description of the conceptual design, the rationale for the algorithm selection and evaluation process before continuing with the logical design and concluding with the implementation. In chapter five, the results are analysed and the model is compared to existing models. The findings here are discussed in the next chapter which also discusses some other issues and thoughts that have surfaced. Finally, the last chapter provides a brief summary of the work, makes conclusions and provides a few suggestions for future research.
2. Literature Review

2.1 Establishing context

Authentication using passwords has been around almost since the dawn of computers and continues to be a very common means of authenticating users seeking access to organizational resources, services or information. Because of its widespread use, it has been subjected to a number of different attacks, including online attack, offline dictionary attack, specific account attack and inversion attack (Chakraborty & Mondal, 2016; Stallings & Brown, 2015). Although alternative methods of authentication have been developed that are considered more secure, e.g. token-based or two-factor authentication or the use of biometric characteristics, using passwords as a means of granting users access to a system is likely to continue well into the future (Herley & Van Oorschot, 2012). Neglecting to appreciate the fact that password use will not be extinct in the near future has discouraged much research into the matter, thus failing to further the security (ibid.).

Having established that the use of passwords will continue, enhancing the overall security of the passwords can be approached from different perspectives. Passwords themselves can be made more difficult to break by making them more complicated, something which in an organizational context normally is guided by passwords policies and guidelines. In a study from 2016, the authors indicate that users have become better at choosing more complex passwords (Shen et al., 2016). User awareness on the importance of choosing strong passwords, and the potential consequences of failing to do so is another approach. Lists of user passwords, either for online accounts or in corporations, need to be adequately protected from disclosure. Not only need access to the lists be secured, the information in the lists must be secured as well, something which is typically done by storing hashed copies of the passwords along with a salt value (Stallings & Brown, 2015). Still, as indicated above, these measures are not always sufficient. Yet another approach to thwart compromise is to make an attacker mistakenly identify fake objects for real ones, in this case passwords.

From time to time, administrators have been known to set up fake accounts, so called honeypot accounts, that set off an alarm when accessed, indicating that an intruder is in possession of account and password information (Juels & Rivest, 2013). However, the intruder may be able to call the bluff and correctly identify these as fake accounts and thus avoid detection (ibid.). The idea to use false passwords and mix them with real ones in an attempt to “hide” the real ones was first introduced by Bojinov et. al. in 2010, using a method referred to as Kamouflage (Bojinov, Bursztein, Boyen, & Boneh, 2010). The method does not protect the password list per se, rather the idea is to make it more difficult for anyone in possession of the list to distinguish real passwords from fake ones. Moreover, the setting for the study is on the protection of a client-side password manager, for example found on a laptop. The fake passwords used to thwart attacks that in some way are derived from the real passwords are the so called honeywords.

A honeyword is a false password that is associated with a specific user account, and whose purpose is to decrease the likelihood that an attack executed by an adversary in possession of a stolen password list remains undetected (Erguler, 2016; Juels & Rivest,
2013). Honeywords can be viewed as an extension of the use of honeypot accounts mentioned above, that administrators can set up in an attempt to detect intruders; the obvious difference being that instead of just having a few fake accounts, every account will have numerous passwords, where only one of them is genuine and attempts to use any non-genuine password will be detected (Juels & Rivest, 2013). The chances of an intruder avoiding detection is thus substantially lower when honeywords are employed as a defence mechanism as opposed to using honeypot accounts.

In connection to honeywords, the two terms sugarword and sweetword are sometimes encountered. A sugarword is simply the correct password whereas sweetwords are all the potential elements of a password list, i.e. honeywords and the sugarword (Chakraborty & Mondal, 2017; Erguler, 2016). When honeywords are employed as a defence mechanism Chakraborty & Mondal refer to this as Honeyword based authentication technique, HBAT, which can be regarded an extension of traditional password authentication schemes (2017).

2013 saw the light of an important study by Juels & Rivest where several different methods of generating honeywords are suggested, some of which are introduced and discussed in the following subsection (Juels & Rivest, 2013). In the study, the authors make the distinction between legacy UI-procedures and modified UI-procedures, where UI is short for user interface. While the latter generation procedure involves modifying the user password selection process by for instance adding a randomly generated number to the chosen password that the user needs to remember as part of the new password, the former procedure contrastingly makes no changes at all. From this, it is clear that the second procedure puts some extra burden on the user whereas the first one has no impact on usability. For the purpose of this study, the goal is to not impact usability negatively and so the proposed artefact will belong to the first category, albeit in a slightly different manner as shall be demonstrated in the coming chapters.

2.2 Existing algorithms for the generation of honeywords

In this section, previously proposed algorithms for the generation of honeywords from existing literature are briefly summarized and contrasted. The different algorithms discussed are shown below:

- Chaffing by tweaking
- Chaffing-with-a-password-model
- Tough nuts
- Take-a-tail
- Hybrid
- Modelling-syntax
- Modified tail
- Close-number-formation (CNF)
- Caps-key

---

3 (Juels & Rivest, 2013)
4 (Bojinov, Bursztein, Boyen, & Boneh, 2010)
5 (Chakraborty & Mondal, 2016)
Leveraging an Active Directory for the generation of honeywords

- Storage-index
- Paired Distance Protocol, PDP

It should be noted that the different methods discussed here are done so in a breve manner and the interested reader who seeks to gain a deeper understanding is advised to turn to the original sources for more thorough explanations.

**Chaffing by tweaking**

In this method, honeywords are derived from a user’s password by tweaking different character positions of the password. The characters should be of the same type, i.e. a digit should be substituted for a digit, a special character for a special character and so on. The number of characters that are to be substituted are determined beforehand. A variation is called *chaffing-by-tail-tweaking* where the password is divided into a head and a tail. The characters making up the tail is then substituted for other characters, again pertaining to the same type. An example of chaffing by tweaking, the password in bold followed by the honeywords:

| pass.word12 | pass:word84 | pass-word65 | pass_word99 |

**Chaffing-with-a-password-model**

This method uses a list of actual passwords that serve as a basis for the construction of honeywords. Note that it does not need to be passwords that are used in the system in question but that it may be passwords derived from online sources or other literature. If online resources are used, it should preferably be used in combination with some other resource(s) to avoid the chance that the adversary has access to the same material, thus facilitating honeywords detection. The characters making up the passwords are subsequently substituted for other characters to produce the honeywords. The number of characters to tweak, their position and whether they should be substituted for characters belonging to the same group or not can vary according to taste.

**Tough nuts**

A tough nut is a deliberately complex honeyword that should be impossible or close to impossible to crack by an adversary. The idea is that the list will not be 100% cracked, leaving certain positions unreadable. As the list will in effect only be a partial one instead of a complete one, this will cast additional doubt as to whether the un-cracked entries are real passwords or honeywords, causing further ambiguity on the adversary’s part. The tough nuts need not even be passwords, but just random strings. For the best effect, the distribution and number of tough nuts within a password list should be random.

---

6 (Erguler, 2016)
7 (Chakraborty & Mondal, 2017)
Take-a-tail

This method resembles the previously described chaffing-by-tail-tweaking, but with an important distinction in that the tail is determined by the system and not the user. As a result, it should be noted that since this method relies on the addition of a randomly generated string of characters to the password chosen by the user, some extra strain is put on the user to remember the new password, thus decreasing usability to some degree.

When a user is prompted for a new password, he suggests one in the usual manner. The system then appends a random sequence of characters to the user’s proposed password. The concatenation of the strings is the resulting user password. An example of the process:

1. User enters a new password: password
2. System generates a random tail: A123
3. The new password is generated: passwordA123

Honeywords can subsequently be generated from the newly created password by tweaking the contents of the tail, for example:

passwordA123 passwordD804 passwordg659 password9W@4

Hybrid

The hybrid method as the name implies, is the combination of several honeyword generation methods into a hybrid one, whereby different methods’ advantages and strengths can be leveraged and utilized in the creation of an enhanced algorithm. Juels and Rivest provide the following example of chaffing-with-a-password-model and chaffing-by-tweaking-digits (2013): First, chaffing-with-a-password-model is employed on a password $p$ to generate a set $\alpha$ of seed sweetwords $W$. Second, chaffing-by-tweaking-digits is applied to each seed sweetword to generate $\beta$ tweaks, resulting in a set $W$ of sweetwords where $k = \alpha \times \beta$. Lastly, $W$ should be permuted to create additional honeywords. An example is provided for $k = 12, \alpha = 3, \beta = 4$, the password is pear215.

pear215 frog741 tree556
pear891 frog801 tree498
pear411 frog956 tree201
pear296 frog834 tree347

The authors conclude that the hybrid method is the most effective one in the generation of honeywords as it delivers great flatness and is resistant to DoS attacks (Juels & Rivest, 2013).
Modelling syntax

This method uses a three-stage model consisting of the steps tokenization, validation and generation. The name of the model stems from the idea to use the same syntax for the honeywords as that of the real passwords. In the first step, the real passwords (sugarwords) are converted into a set of rules, or rather parsed into tokens, each representing a different syntactical element. Tokens are typically of the type, word, number or special character. In the validation step, the tokenization process is confirmed by the system and tokens that are not dictionary words are flagged, indicating that user review is required. At this point, the user is given an opportunity to remove any word that is considered personal, e.g. names and dates. In the final step, the honeywords are generated using the tokens and rules derived in the preceding steps. The actual token substitution is dependent on a dictionary that can be selected according to the application’s deployment environment or other specific requirements. The dictionary is then inputted to the generation algorithm.

An example of the modelling syntax:

The password `guitar@store95` is tokenized as `W_6 | S_1 | W_5 | D_2`, i.e. a 6-letter word, a 1-character special character, a 5-letter word and a 2-digit number. To generate the honeywords, the tokens are substituted with tokens of the same type, for example yielding the following honeywords:

```
guitar@store95   dinner&table47   banana?apple12   silver£shelf55
```

Had one of the words been flagged as not being part of the utilized dictionary, e.g. a name like Marty, the system can as part of the validation step inform the user that this word is not suitable for generating honeywords as this will make the actual password stand out from the honeywords, thus diminishing its effectiveness.

Moreover, as noted by Juels and Rivest, honeymoons generated using this model unlike those constructed from the related chaffing-with-a-password-model described previously, are dependent on the real password (2013).

Modified tail

In this method the user chooses some extra information that is appended to the end of the password, i.e. the tail. The characters are chosen from a set ($S$) of characters, that should be of $S - 1$ length. The system then adds the remaining character from the set to form the complete password. The characters need to be selected in a manner so as to avoid obvious keyboard patterns, like for example qwerty or plok. For the generation of honeywords, different combinations of the characters from $S$ are employed to make up the tail. The tail is then appended to the password to make up the complete password. As is evident from previous algorithms based on some usage of the tail concept, it is only the tail part of the password that is tweaked whereas the password itself remains unaltered throughout the process.
An example of this method is as follows. If the user selects as password apple and for the tail @* and the set $S$ is @*#, then the sugarword is apple@*#. Note that the user only needs to supply the password apple@* as the “#” is supplied by the system. The possible honeywords are thus the remaining combinations of the characters from the set $S$ appended to the head:

apple@*# apple@** apple#@* apple#@ apple#@ apple#@#

**Close-number-formation**

Close-number-formation or CNF uses numbers from the original password that are in close proximity to these to generate the corresponding honeywords. A password of July2005 can thus be modified to generate honeywords such as July2006. The algorithm employs two sets $num = \{1, 2, 3\}$ and $sig = \{+, −\}$. The starting digit is then first retrieved before being tweaked and given another value. The process is then repeated to tweak the remaining digits as desired. Possible $k$ sweetwords for $k = 6$ can thus be as follows:


By only making small shifts in the numbers, there is an increased likelihood that the change does not make them distinguishable from the original number. If the original number represents a year, like 2005 in the example above, just substituting the digits randomly could yield the number 5671 which is clearly not a date and is thus quite distinct from the original password, making it easier for an adversary to detect. This is an important advantage as it is quite common for users when selecting passwords to incorporate numbers that are meaningful to them, like birthdays or other important dates (Erguler, 2016; Shen et al., 2016). This is further discussed in section 2.5 on passwords below.

Some digit patterns cannot be disguised accurately using CNF and the authors suggest a technique labelled pre-processing to remedy this (Chakraborty & Mondal, 2016). Two categories of numbers are discerned; repetition of digit $d$ for $d$ times and repetition of digit $d$ for $d' \times$ times. An example of the first category would be 4444 and of the second 555. Honeywords examples from these where $k = 7$:

<table>
<thead>
<tr>
<th>4444</th>
<th>22</th>
<th>5555</th>
<th>333</th>
<th>88888888</th>
<th>666666</th>
<th>7777777</th>
</tr>
</thead>
<tbody>
<tr>
<td>555</td>
<td>111</td>
<td>333</td>
<td>999</td>
<td>444</td>
<td>222</td>
<td>888</td>
</tr>
</tbody>
</table>

**Caps-key**

This method utilizes variations in case, i.e. lowercase and uppercase characters to generate different honeywords from the original password. The user chooses $m$ uppercase letters from the password of length $n$. The number of possible honeywords can be calculated according to the formula $\binom{n}{m} - 1$. For $m = 3$ and $n = 7$ this yields 34 possible variations of a single password. To generate $k$ sweetwords for $k = 5$ for the password DruMkiT, possible honeywords are shown below:
The author’s prime motivation for this method is to decrease the storage overhead that is typically associated with the use of honeywords (Erguler, 2016). As opposed to storing the generated honeywords in the password file as is the norm, the idea is to utilize already existing passwords to resemble honeywords, thus reducing the need for increased storage as no new honeywords are added. This means that contrary to all other honeyword methods described, no new honeywords are actually created; the method instead relies on existing passwords to appear as honeywords for other accounts. The setup is as follows:

For every account, the existing password indexes are assigned in a random fashion to a newly created user account. These indexes are referred to as honeyindexes. The account is given a random index number and together with a hash of the correct password and correct index is stored in a list. At the same time, the username together with an integer set consisting of the correct index and the created honeyindexes are stored in a different list. The effect is that when an adversary analyses the two compromised lists, it becomes evident that each username corresponds to several numbers in the form of sweetindexes, each pointing to a real password. Thus, it is no longer clear which password belongs to which user and the submission of an incorrect one will trigger the alarm as per usual. Apart from just reducing the storage needed for the password list, another contribution is that since real user passwords are being used as honeywords, an adversary has no chance of detecting the honeywords as being fake as in effect no fake passwords have been used.

Paired Distance Protocol

The Paired Distance Protocol or PDP, is like most other algorithms dependent on three parts; username, password and a password-tail. The password and the tail are chosen by the user and although the latter can be reused for other accounts in other systems, there is still some extra information that needs to be remembered, thus affecting usability, albeit to a slight degree. The tail is constructed from a Honey Circular List (HCL) that consists of randomly distributed characters from a set of alphabets and numerals, see Figure 1 right. The length of the tail should be \( l > 1 \) and the default value is 2. The paired distance \((PD)\) is calculated as the distance between two elements, counting clockwise. For example, the distance between “v” and “5” is 3. In the password file \(F\), username, password and the corresponding \(PD\) are stored. The honeychecker stores only the username and the first element of the chosen password-tail.

\[\text{Figure 1 – Honey Circular List (Chakraborty \\& Mondal, 2017, p. 159)}\]
For \( l = 2 \) there are 36 different possible password-tails that can be regarded as virtual sweetwords. An adversary is thus faced with 36 possibilities when attempting a login, only one of which is the genuine password.

The authentication process for PDP is a bit different from the methods discussed previously. If an adversary is in possession of \( F \), he supplies the username and the password but has to make a guess for the password-tail. The system verifies the login information and since the correct username and its associated password have been entered proceeds to the next step. In this step, the system tries to validate the submitted tail by calculating the \( PD \) from the supplied tail and comparing this with the stored value. If there is a match, the system initiates communication with the honeychecker. The username and first character of the tail are then sent for verification by the honeychecker. Should any of the described authentication steps fail, an alarm is triggered and compromise is subsequently detected.

### 2.3 Metrics for evaluation

To be able to compare and evaluate the different algorithms which is the topic of the next subsection 2.4, it is necessary to first review the metrics and variables typically associated with performance evaluation. This is the purpose of this subsection. The variables are grouped into different sections according to Chakraborty & Mondal (2016).

#### 2.3.1 Security standards

**Flatness**

If the number of honeywords for each user is represented as \( n \), then every user will have \( n + 1 \) passwords stored in the password list. As can be seen from the equation showing attacker success probability \( \frac{1}{(n+1)} \), the success ratio diminishes depending on \( n \) (R. Wang, Chen, & Sun, 2016), i.e. the more honeywords used, the greater the chance that a honeyword is chosen and an intrusion attempt is detected.

The property of flatness refers to the fact that among a distribution of sweetwords there is an equal chance of an adversary picking a honeyword as the sugarword, i.e. the correct password should blend in perfectly among the honeywords. A generation procedure is said to be **perfectly flat** if the attacker has no advantage in guessing the correct password in a distribution of sweetwords (Chakraborty & Mondal, 2017). The probability of guessing the correct password from a population \( W \) is \( \frac{1}{k} \) where \( k \) is the number of elements in \( W \), and conversely \( \frac{(k-1)}{k} \) of hitting a honeyword. For \( k = 4 \), there is a 25% chance of selecting the correct password, thus as noted above, success is dependent on \( k \) (Erguler, 2016). For values slightly greater than \( \frac{1}{k} \), the algorithm is considered **approximately flat** (Juels & Rivest, 2013). Should the algorithm used for the generation of honeywords fail to be flat enough, the correct password will become more easily discernible for an adversary, thus undermining the technique’s usefulness (Erguler, 2016).
DoS resistance

The intention of using honeywords is to add an additional layer of security in an effort to strengthen the security of password-protected systems. An intrusion attempt by an adversary should preferably be thwarted before he gets access to the system. Using honeywords provides a means to detect a password list breach and subsequent login attempts, resulting in the triggering of an alarm as discussed previously. An adversary can, however, leverage this fact to mount a Denial of Service attack against the system, effectively denying authorized users access to necessary resources. Depending on the security policy in place, this can have serious implications for the users that are dependent on the system. At two ends of the spectrum are the light and heavy security policies, which in this context refer to the blocking of a single account from which a honeyword has been submitted to blocking every available account respectively (Chakraborty & Mondal, 2017). A strict policy might enforce a global password reset affecting all users of a system in response to a single submitted honeyword (Juels & Rivest, 2013).

A DoS attack can be initiated by an attacker in possession of the correct password for an account by guessing the honeywords and intentionally submitting these to trigger the alarm (Chakraborty & Mondal, 2016), leading to the subsequent lockdown of accounts according to the security policies in place (Juels & Rivest, 2013). With the information from a single password, an attacker can create the illusion of a compromised password file when in fact only a single password has been lost, e.g. by phishing or malware attacks (ibid.). It is also possible that an attacker, possibly an insider, has acquired knowledge of the honeyword generation algorithm and thus the capability to produce honeywords from existing passwords (Erguler, 2016).

DoS resistance is an algorithm’s ability to withstand attacks, if honeywords can be easily guessed, then that algorithm is said to have weak DoS resistance (Erguler, 2016). Different methods are more or less susceptible to this type of attack and this is discussed in section 2.4 below.

Multiple system vulnerability

If the same honeyword generation algorithm is used for several systems, different honeywords will be generated from the same password as the output from the algorithm yields different results at runtime. This happens when a user who is registered in two systems has the same password on both systems (Juels & Rivest, 2013). An adversary who manages to obtain the password files from the two systems can perform what is known as an intersection operation in order to reveal the real password from a user (Chakraborty & Mondal, 2016). The two systems need not both employ honeywords for being susceptible to this type of attack; if system A uses honeywords but system B does not, an attacker in possession of both systems’ password files can take the sweetwords for a common user and submit these to system B (Erguler, 2016). Any honeyword submitted to B will simply inform the attacker that an incorrect password has been issued while the correct password will, of course, grant access, and can subsequently be used to gain access to system A as well (ibid.).
This weakness is known as Multiple System Vulnerability or MSV and most honeyword algorithms offer weak protection against the vulnerability (Chakraborty & Mondal, 2016). Thus, protection against MSV means that should a user’s account be compromised in several systems, it is not possible for an attacker to immediately recover the password (Juels & Rivest, 2013).

### 2.3.2 Usability standards

The impact on usability is determined by evaluating the three parameters system interference, memorability stress and typo safety (Chakraborty & Mondal, 2016). The previously mentioned division by Juels & Rivest into modified UI-procedures and legacy UI-procedures help distinguish methods from having or having no impact on usability respectively (2013). The parameters are discussed next.

#### System interference

System interference is the property of whether a user is forced by the system to remember some extra information or not (Chakraborty & Mondal, 2016), i.e. if the honeyword algorithm requires that some extra system-generated information is appended to the user’s password, e.g. take-a-tail or modified tail (Chakraborty & Mondal, 2017). This is thus corresponding to the legacy UI distinction made by Juels & Rivest, where a method that falls into this category, in addition, imposes no directions or restrictions regarding password choice or even informs the user that honeywords are being used (2013). Another apparent advantage with the methods from this category is that since it is completely transparent to the user, it is possible to change the honeyword generation algorithm without the need to alter the UI or inform the users (ibid.).

Algorithms can belong to one of the three different classes of high, low or no system interference (Chakraborty & Mondal, 2017). The first class has the user remember different bits of information for different accounts; for the second class, the same information can be used for several different accounts and it may be of the user’s own choice; finally, in the third class, the user need not remember any additional information (ibid.).

#### Memorability stress

The parameter memorability stress is closely related to system interference discussed above, and as the name implies refers to what degree a method that imposes system interference exercises stress on the user (Chakraborty & Mondal, 2016). Two categories can be distinguished; methods that exercise low stress, e.g. caps-key, and methods that cause high stress among the users, e.g. take-a-tail (ibid.). The disadvantages of this latter method become even more pronounced should the user utilize the same password for several accounts in multiple systems as each would produce a different tail that needs to be remembered (Erguler, 2016). A good honeyword generation algorithm should always strive for low stress on memorability (Chakraborty & Mondal, 2016).
Typo safety

A typo safe honeyword generation algorithm is one that results in only a negligible probability that during login, a user who mistakenly mistypes the password instead submits a honeyword (Chakraborty & Mondal, 2016). With an algorithm of low typo safety, there is a greater risk that a user hits a honeyword. A single typo must not lead to a legitimate user being locked out of the system due to an accidental honeyword hit as a consequence of a mistyped password (Juels & Rivest, 2013). Thus, from a usability standpoint, any good honeyword algorithm should always be typo safe (Chakraborty & Mondal, 2017). In addition, should a strict security policy be in place, a single typo has the potential of initiating a global password reset as discussed above (Juels & Rivest, 2013).

Moreover, typo safety is equally relevant for both modified and legacy UI-procedures, as there will always be a chance that a user accidentally types a honeyword regardless of whether the generation algorithm makes the user remember some extra piece of information or not.

2.3.3 Storage cost

At the heart of every honeyword technique is the generation of additional decoy passwords. A direct consequence of this is the increased need for additional storage. Although hard disk storage from an economic viewpoint is not of particular concern these days, it is still considered one of the major drawbacks of the honeyword technique (Chakraborty & Mondal, 2016; Erguler, 2016). For each user account in a system, most honeyword generation algorithms produce \( k - 1 \) honeywords where \( k \) is the total number of sweetwords (Chakraborty & Mondal, 2017). A system of \( n \) users thus produces \( n(k - 1) \) of additional information (ibid.), which for a value of \( k = 20 \) as suggested by Juels & Rivest (2013), and where a size of each hashed element equals 20 bytes, adds up to a total of 380 bytes increase in storage per user (Chakraborty & Mondal, 2017).

2.4 Algorithm comparison and evaluation

In this section, the previously discussed honeyword generation algorithms are evaluated according to the metrics introduced above. Please note that the tables and data presented here are based on the comparisons and evaluations of the methods as conducted by the referenced authors.

In Table 1 below, Juels & Rivest have compiled the results of the methods proposed in the paper, evaluated according to some of the performance metrics (2013, p. 10). Things to note are:

- The storage cost is calculated based on the assumption that \( k - 1 \) honeywords have been generated and is displayed as the total number of hashes, not the cumulative size.
- Tough nuts are most effective when used in combination with other methods.
● Flatness: All of the proposed models can deliver perfect flatness, i.e. \( \frac{1}{k} \), albeit under some conditions. If \( G \) represents the probability distribution of honeywords that have been generated with the chaffing-with-a-password-model and \( U \) is the probability distribution of passwords that adhere to a specific policy determined beforehand, then \( U \approx G \) means that from the point of view of an attacker, the honeywords are distributed much like regular user passwords. \( T(p) \) denotes the sweetwords possible after tweaking the password \( p \).

<table>
<thead>
<tr>
<th>Honeyword method</th>
<th>Flatness</th>
<th>DoS Resistance</th>
<th>Storage cost</th>
<th>Legacy-Ul</th>
<th>Multiple system protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tweaking</td>
<td>( \frac{1}{k} )*</td>
<td>Weak</td>
<td>1</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Password-model</td>
<td>( \frac{1}{k} **</td>
<td>Strong</td>
<td>( k )</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Tough nuts</td>
<td>N/A</td>
<td>Strong</td>
<td>( k )</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Take-a-tail</td>
<td>( \frac{1}{k} )</td>
<td>Weak</td>
<td>( k )</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Hybrid</td>
<td>( \frac{1}{k} ***</td>
<td>Strong</td>
<td>( \sqrt{k} )</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

* if \( U \) constant over \( T(p) \)
** if \( U \approx G \)
*** if \( U \) constant over \( T(p) \) and \( U \approx G \)

In a recent study, the results from Table 1 have been questioned, in particular the flatness parameter (D. Wang et al., 2017). The aim of the study is to quantitatively assess the performance of four of the honeyword methods as proposed by Juels & Rivest (2013). Two types of attackers are distinguished, labelled \( A_1 \) and \( A_2 \) where \( A_1 \) corresponds to the attacker as presumed by Juels & Rivest while \( A_2 \) is considered a more powerful attacker in that he is in possession of the users' personally identifiable information, i.e. PII (D. Wang et al., 2017). Armed with this type of extra information, an attacker has a far greater chance of distinguishing the real password from the honeywords.
In the experimental setup for conducting the evaluation, the authors make use of two different probabilistic password cracking models: Markov\(^8\) and PCFG\(^9\), as well as a third model, simply referred to as List, which basically is a probability distribution of some leaked datasets (D. Wang et al., 2017). The models are used as is by the attacker \(A_1\), whereas attacker \(A_2\) uses them together with some knowledge of user PII. The results are shown in Table 2.

In the paper, it is argued that the metric of flatness is inadequate and two related metrics are introduced, however, the table below still displays the success rate of distinguishing the honeyword from a group of sweetwords as a measure of an algorithm’s strength (D. Wang et al., 2017). The numbers represent the likelihood in per cent that the honeyword is found in a population of sweetwords of size \(k = 20\) in a single guess. A flat algorithm as explained by Juels & Rivest can be expressed as \(\frac{1}{k}\) (2013), which for \(k = 20\) corresponds to 5\%, meaning that there is only a 5\% chance that an adversary can pick the actual password. From the evaluation of the methods whose results are displayed in Table 2 below, it is quite clear that these numbers have been greatly overestimated (D. Wang et al., 2017).

The shaded boxes indicate the top performance among its attacker category and the \textit{Tar} prefix for attacker \(A_2\) is short for target, i.e. it indicates a targeted guessing attack.

<table>
<thead>
<tr>
<th>Honeyword method</th>
<th>Type (A_1) attacker</th>
<th>Type (A_2) attacker</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>List</td>
<td>PCFG</td>
</tr>
<tr>
<td>\textit{Tweak-tail}</td>
<td>35.6</td>
<td>49.0</td>
</tr>
<tr>
<td>\textit{Modelling-syntax}</td>
<td>32.6</td>
<td>43.9</td>
</tr>
<tr>
<td>\textit{Hybrid}</td>
<td>34.6</td>
<td>47.8</td>
</tr>
<tr>
<td>\textit{Simple model}*</td>
<td>34.2</td>
<td>17.1</td>
</tr>
</tbody>
</table>

*Corresponding to Juels & Rivest’s password model

In Table 3 below, adapted from Chakraborty & Mondal (2016, p. 807), some of the methods introduced by Juels & Rivest (2013), are compared with those proposed by the authors. The paper’s main objective is to improve the storage cost and the proposed methods make use of the concept of paired-distance-protocol, PDP, that is elaborated on in their subsequent paper (Chakraborty & Mondal, 2017). The methods introduced

\(^8\) A Markov chain is a model that describes a sequence of events, where the probability of each successive event only depends on the state of the event immediately preceding it.

\(^9\) Probabilistic context-free grammar, a template-based model used on disclosed password datasets to generate password patterns in an attempt to reduce the number of guesses required to crack passwords (Weir, Aggarwal, De Medeiros, & Glodek, 2009).
in this paper all begin with the prefix SO, which stands for storage optimization and are the following: Storage optimized modified-tail (SOMT), close-number-formation (SOCNF), caps-key (SOCK) and pre-processing (SOPP). Apart from displaying reduction in the storage cost, the proposed methods also show some improvements for some of the security parameters.

<table>
<thead>
<tr>
<th>Honeyword method</th>
<th>System interference</th>
<th>Memor ability stress</th>
<th>Typo-safe</th>
<th>DoS-safe</th>
<th>MSV-safe</th>
<th>Flatness</th>
<th>Extra storage cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling-syntax</td>
<td>No</td>
<td>Low</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
<td>(\frac{1}{k^*})</td>
<td>(k - 1)</td>
</tr>
<tr>
<td>Take-a-tail</td>
<td>Yes</td>
<td>High</td>
<td>Yes</td>
<td>Low</td>
<td>High</td>
<td>(\frac{1}{k})</td>
<td>(k - 1)</td>
</tr>
<tr>
<td>Modified-tail</td>
<td>Yes</td>
<td>Low</td>
<td>Yes</td>
<td>Low</td>
<td>High</td>
<td>(\frac{1}{k})</td>
<td>(k - 1)</td>
</tr>
<tr>
<td>CNF</td>
<td>No</td>
<td>Low</td>
<td>No</td>
<td>Low</td>
<td>High</td>
<td>(\frac{1}{k^*})</td>
<td>(k - 1)</td>
</tr>
<tr>
<td>Caps-key</td>
<td>Yes</td>
<td>Low</td>
<td>No</td>
<td>Low</td>
<td>Moderate</td>
<td>(\frac{1}{k})</td>
<td>(k - 1)</td>
</tr>
<tr>
<td>Chaffing-by-tweaking**</td>
<td>No</td>
<td>Low</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
<td>(\frac{1}{k^*})</td>
<td>(k - 1)</td>
</tr>
<tr>
<td>Pre-processing</td>
<td>No</td>
<td>Low</td>
<td>Yes</td>
<td>Low</td>
<td>High</td>
<td>(\frac{1}{k})</td>
<td>6 and 9</td>
</tr>
<tr>
<td>SOMT</td>
<td>Yes</td>
<td>Low</td>
<td>Yes</td>
<td>High</td>
<td>High</td>
<td>(\frac{1}{k^*})</td>
<td>1</td>
</tr>
<tr>
<td>SOCNF</td>
<td>No</td>
<td>Low</td>
<td>No</td>
<td>Low</td>
<td>High</td>
<td>(\frac{1}{k^*})</td>
<td>Null</td>
</tr>
<tr>
<td>SOCK</td>
<td>Yes</td>
<td>Low</td>
<td>No</td>
<td>Low</td>
<td>High</td>
<td>(\frac{1}{k})</td>
<td>Null</td>
</tr>
<tr>
<td>SOPP</td>
<td>No</td>
<td>Low</td>
<td>Yes</td>
<td>Low</td>
<td>High</td>
<td>(\frac{1}{k})</td>
<td>1 and 2</td>
</tr>
</tbody>
</table>

* Flateness associated with some conditions
** Chaffing-by-tweaking-digits (CTD)

Table 3 – Comparative analysis of methods (Chakraborty & Mondal, 2016, p. 807)

Table 4 and Table 5 below, adapted from Chakraborty & Mondal, show the evaluation results from several of the methods in comparison to each other and the method of PDP as proposed by the authors (2017, pp. 166–167). Please note that the tables have been slightly modified to improve readability.

Some remarks on the notation used: \(H^P\) is the number of possible honeywords from a given password and \(H^S\) is the actual number of honeywords produced by an algorithm, which eventually would lead to \(H^S = K - 1\). A large value is denoted by \(L^V\); \(T^S\) simply means tough nuts.
For the quantitative analysis, the authors provide a framework in order to analyse the security parameters; a few remarks and requirements for safety against certain vulnerabilities and attacks are as follows (Chakraborty & Mondal, 2017, p. 164):

- Requirement for DoS resiliency: $\mathcal{H}^P \gg \mathcal{H}^S$
- Requirement for MSV safety: $\mathcal{H}^P \approx \mathcal{H}^S$
- Requirement for typo safety: $\mathcal{H}^P \gg \mathcal{H}^S$
- Requirements for both DoS resiliency and MSV safety: As the requirements stated above are contradictory and cannot be satisfied simultaneously, the authors’ proposed resolution is something they call collaborative security, which in essence means that a system whose password list has been compromised sends a security message to the other participating systems, notifying them of the breach.

As discussed before, flatness can be described as $\frac{1}{k}$. For the $\mathcal{H}^S$ parameter, this can be represented as $\frac{1}{(\mathcal{H}^S + 1)}$.

<table>
<thead>
<tr>
<th>Honeyword method</th>
<th>$\mathcal{H}^P$</th>
<th>$\mathcal{H}^S$</th>
<th>Flatness</th>
<th>Storage overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tweaking</td>
<td>99</td>
<td>19</td>
<td>0.05*</td>
<td>380</td>
</tr>
<tr>
<td>Modelling-syntax</td>
<td>$L^U$</td>
<td>19</td>
<td>0.05*</td>
<td>380</td>
</tr>
<tr>
<td>Take-a-tail</td>
<td>999</td>
<td>19</td>
<td>0.05**</td>
<td>380</td>
</tr>
<tr>
<td>Tough nuts</td>
<td>$L^U$</td>
<td>$T^ST^S(&lt;\mathcal{H}^S)$</td>
<td>N/A</td>
<td>$20 \times T^S$</td>
</tr>
<tr>
<td>CNF</td>
<td>114</td>
<td>19</td>
<td>0.05*</td>
<td>380</td>
</tr>
<tr>
<td>Caps-key</td>
<td>28</td>
<td>19</td>
<td>0.05**</td>
<td>380</td>
</tr>
<tr>
<td>Modified-tail</td>
<td>6</td>
<td>6</td>
<td>0.017**</td>
<td>100</td>
</tr>
<tr>
<td>Storage-index</td>
<td>$L^U$</td>
<td>19</td>
<td>0.05*</td>
<td>84</td>
</tr>
<tr>
<td>PDP</td>
<td>1259</td>
<td>35</td>
<td>0.028***</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4 – Quantitative comparison of methods (Chakraborty & Mondal, 2017, p. 166)

* if $U \approx G$ (as above)
** unconditionally
*** if randomness of a password tail is high
2.5 Passwords

The principal aim of this subsection is to review existing literature in an attempt to discover how users typically go about constructing passwords. Particular emphasis will be on determining common password characteristics and to examine passwords that are based on personal information, as it is this type of information that is most reminiscent of that found in an AD. The findings will play an integral part in the subsequent construction of an algorithm for the generation of plausible honeywords.

2.5.1 Introduction

Passwords are by far the most widespread means of authentication in use today and have been so for about half a century (Bonneau et al., 2015). Moreover, all evidence points to that they will continue to be around in the foreseeable future despite their proven weaknesses and shortcomings (Bonneau et al., 2015; Herley & Van Oorschot, 2012; Li, Wang, & Sun, 2017). Some reasons that can be attributed to this include: passwords are inexpensive when compared to the use of physical devices, e.g. smart cards or fingerprint scanners; a password system is fairly easy to deploy and administer; the use of single sign-on for authentication to multiple systems utilizing other authentication techniques creates a single point of failure; a password-protected account can be accessed from anywhere by just using a web browser; and they are easily understood by the users (Herley & Van Oorschot, 2012; Stallings & Brown, 2015).

<table>
<thead>
<tr>
<th>Honeyword method</th>
<th>DoS-safe</th>
<th>MSV-safe</th>
<th>Typo-safe</th>
<th>System interference</th>
<th>Memorability stress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tweaking</td>
<td>0.51</td>
<td>0.98</td>
<td>0.8</td>
<td>No</td>
<td>Low</td>
</tr>
<tr>
<td>Modelling-syntax</td>
<td>≈ 1</td>
<td>𝜖*</td>
<td>≈ 1</td>
<td>No</td>
<td>Low</td>
</tr>
<tr>
<td>Take-a-tail</td>
<td>0.94</td>
<td>1**</td>
<td>0.98</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Tough nuts</td>
<td>≈ 1</td>
<td>𝜖*</td>
<td>≈ 1</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>CNF</td>
<td>0.57</td>
<td>0.97</td>
<td>0.83</td>
<td>No</td>
<td>Low</td>
</tr>
<tr>
<td>Caps-key</td>
<td>0.002</td>
<td>1</td>
<td>0.29</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Modified-tail</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Storage-index</td>
<td>≈ 1</td>
<td>𝜖*</td>
<td>≈ 1</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PDP</td>
<td>0.91</td>
<td>1</td>
<td>0.97</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 5 – Quantitative comparison of methods (Chakraborty & Mondal, 2017, pp. 166–167)

*The probability 𝜖(> 0) infers a negligible value

**MSV-safe for P becomes 1 due to cumulative changes in passwords for each account
Moreover, there is a lack of consensus among the security community as to the passwords’ level of security as well as suitable metrics for establishing the security (Bonneau, 2012). In particular, the enormous gap between the theoretical password space, i.e. all possible passwords, and those actually used in a real password distribution makes it especially important to choose relevant metrics so that adequate estimates of strength can be established (Bonneau, 2012; Bonneau et al., 2015). Past failure to realize that users do not choose their passwords in a manner similar to that of a random password generator has led to overestimates of security within a particular distribution (Bonneau et al., 2015) thus, to study how users actually choose their passwords is crucial in order to be able to draw educated conclusions.

It might seem tempting to enforce a policy that stipulates that users choose extremely complex passwords. This can, however, have an opposite effect as in order to remember the passwords, users may feel the need to write them down or to store them in an unsafe manner (Grassi et al., 2017). However, choosing a complex password and having to write it down normally provides stronger security than choosing a simple one as most attacks are conducted over the network (Cheswick, 2013).

As the intended environment of the proposed solution is in an organizational or corporate setting, the primary attack scenario is that of an offline attack, which requires more resistance than do the online attack (Bonneau et al., 2015; Herley & Van Oorschot, 2012), i.e. passwords need to be more resistant to withstand cracking efforts. The strength of the passwords is typically dependent on the type of information or resources present on the system and the level of security required (Herley & Van Oorschot, 2012) and is normally regulated in policies (Bonneau et al., 2015). However, it is not uncommon that the only reason these policies are in place, is so that the organization can be compliant with certain regulations, security audits or simply to adhere to best practices (Herley & Van Oorschot, 2012).

### 2.5.2 Characteristics in user password construction

If a user is given no restrictions when choosing a password, for the sake of convenience, an easily guessable one is oftentimes chosen that can include or be based on things such as their street name, name or a common dictionary word (Das, Bonneau, Caesar, Borisov, & Wang, 2014; Stallings & Brown, 2015; Taneski et al., 2014). The use of dates, either as part of or as the complete password is another source of inspiration when constructing passwords (Veras, Thorpe, & Collins, 2012). The user’s first language is the natural choice for passwords that are based on dictionary words (Bonneau, 2012; Li et al., 2017). When studying the semantic patterns of passwords, authors concluded that a great number of passwords contain concepts relating to love, food, money, animals, sexual terms and profanity (Veras, Collins, & Thorpe, 2013). Using the actual login name as a password is unfortunately not uncommon either (Shen et al., 2016) and in a paper from 2013, authors report that more than half of the sampled sites allow passwords and usernames to be identical (Castelluccia et al., 2013). Furthermore, relying on keyboard patterns to create seemingly complex passwords that comply with even the most stringent of policies is another characteristic that can be exploited when cracking passwords (Schweitzer, Boleng, Hughes, & Murphy, 2011). Domain information is yet another piece of information that users might employ in their passwords, especially for online accounts. In this context, domain refers to an Internet
domain, and users can choose to include either part of or the full address information in the password for a particular site (Malone & Maher, 2012), making it easier to associate the correct password with its corresponding site (Li et al., 2017).

Whether the nature of the data itself, i.e. its sensitivity and importance to the user can or cannot be linked to password strength in terms of password length and composition has been debated. Zviran & Haga concluded that for more valuable data, a longer password is not chosen nor are more non-alphanumeric characters utilized (1999). In a report from 2007, the authors came to the conclusion that users tended to employ stronger passwords for corporate accounts than for other sites, although this may have been attributed to the fact that the corporate site mandated that stronger passwords be used (Florencio & Herley, 2007). Moreover, recent studies show that password strength has improved in later years due to the public’s increased security awareness (Shen et al., 2016). In addition, as the number of password-protected accounts or systems that a normal user comes into contact with is non-negligible, it is not surprising that the same password is employed for several accounts and that it for sake of memorability is comprised of meaningful data such as names and dates (J. Campbell, Ma, & Kleeman, 2011). Moreover, studies of password distributions from different leaked datasets reveal apparent similarities between the sets, i.e. a password that is common in one list is likely to also be popular in another list, something which of course can be leveraged by an adversary (Malone & Maher, 2012).

According to Shen et al., password choice can generally be divided into four different categories; single-meaningful data, combo-meaningful data, pronounceable and random characters, where passwords belonging to the second category are by far the most common according to their findings (2016). Their study also shows that users are prone to use meaningful data when constructing their passwords as well over 80% of the passwords studied belong to the first two categories (ibid.). This is confirmed by Zviran & Haga in an earlier report, where close to 80% of the passwords were composed by either a single or a combination of meaningful details (1999).

To ensure that users refrain from using the weakest and most common passwords, user password selection should be constrained by either using a system that automatically filters out poor passwords (Taneski et al., 2014) or preferably, a “black list” consisting of previously leaked, weak or otherwise unsuitable passwords against which user-proposed passwords are compared and if necessary, subsequently rejected (Bonneau et al., 2015; Grassi et al., 2017). Like previously mentioned, user password composition is normally guided by policies that have been adopted by the company. Password policies can be considered a subset of authentication policies which normally comprise the largest set of policies in an information system environment due to the complexity and variety of the systems (Rhodes-Ousley, 2013). Examples of password policies include: privacy, construction, reset, expiration and reuse (ibid.). A typical password composition policy might state that passwords be of minimum eight characters in length, need to have at least one uppercase character and include at least one special character. As the composition policy is directly correlated to usability, i.e. the stricter the policy the harder it may be to come up with and remember the passwords, it is no surprise that sites who face competition like shopping sites or social media tend to impose weaker policies than do for example universities (Bonneau et al., 2015; Cheswick, 2013; Das et al., 2014). Whether the policies actually contribute to
enhancing the security is another matter. Campbell et al. argue that it is an understudied area and conclude in their report that restrictive password composition policies do improve password resistance against brute force attacks utilizing dictionary words but that the use of meaningful information was not impacted nor was the tendency to reuse passwords across multiple systems (2011).

When analysing password composition, evidence suggests that passwords comprised of lowercase letters alone are by far most common (Florencio & Herley, 2007), and in particular those that only contain letters (Shen et al., 2016). Moreover, reluctance by users to include non-alphanumeric characters can be clearly discerned (Bonneau, 2012). Using passwords containing numbers only is also quite common; in a report from 2016, authors concluded that the figure was over 45% (Shen et al., 2016), although the high number may at least in part be attributed to the fact that the dataset studied was of Chinese origin, where the use of numbers is generally more frequent (Li et al., 2017). Previously leaked password lists also bear evidence to this fact, as passwords such as “123456” are in the top ten of most frequently used passwords in several different lists (Malone & Maher, 2012). When asked to introduce a single digit into a password, users choose the number “1” in about half of the cases (Weir, Aggarwal, De Medeiros, & Glodek, 2009). When including special characters in passwords, characters such as “.”, “!” and “@” are far more common than for instance “|” or “}”, because they are used more frequently in regular typing and considered to be more convenient (Shen et al., 2016). In addition, when users are prompted to include special characters by administrative policies, there is a striking tendency to do so in predictable places; the initially chosen password “password” would for example typically be transformed to “Password1.” when forced to include a capital letter, a number and a special character (Grassi et al., 2017).

2.5.3 Personal information in passwords

In spite a general recommendation to not use dictionary words or information that can be associated with the account owner, users tend to ignore this for various reasons (Taneski et al., 2014). Password composition policies can many times be contradictory and thus difficult to adhere to: do not choose meaningful passwords and do not write them down but be sure to change them frequently (Herley & Van Oorschot, 2012); refrain from using the same password on different systems and ensure that each is sufficiently complex (J. Campbell et al., 2011; Cheswick, 2013); do not use dictionary words and do not include meaningful information (J. Campbell et al., 2011). In addition, it is still not uncommon that the rules are not clearly conveyed to the user making it even more difficult to come up with a password that satisfies the criteria (Cheswick, 2013).

That the inclusion of personal information in passwords reduces the passwords’ strength and makes them less effective to withstand attacks by as much as up to a 30% increase in cracking success, is established in a 2013 report (Castelluccia et al., 2013). This is confirmed by Li et al. who show in their experiments that the possession of personal information is beneficial both to cracking speed and overall success rate (2017). Moreover, acquiring a user’s personal information has become increasingly easier as more users tend to browse Online Social Networks (OSNs) and deliberately
divulge personal information, (Krishnamurthy & Wills, 2010) that in turn can be exploited by malicious actors with criminal intent.

In a study from 2004, authors concluded that over 80% of users include either themselves, family or friends when designing passwords where names, be them proper ones or nicknames, are most frequent (Brown, Bracken, Zoccoli, & Douglas, 2004). When the authors studied a leaked Chinese password dataset from an online railway ticket booking system, they were able to examine the use of personal information as the dataset also contained information such as the user’s name and ID number (Li et al., 2017). They found that more than 60% of the total amount of passwords contained personal information and that account name (username used to log in to the system), birthdate and name were the most common types used with a frequency of over 20% followed by email address that accounted for over 12% (ibid.). At the bottom of the table were ID number (equivalent to Social Security Number, SSN) and cell phone number, both scored a usage ratio of less than 3% (ibid.). This is consistent with the findings by Castelluccia et al. who concluded that usernames, first names and birthdays were the most common PII attributes, with a greater incidence than for example location, friends, work and education (2013). Furthermore, it should be noted that personal information might not always be outright identifiable, a security-conscious user may well choose to mangle it a bit in an attempt to make it less conspicuous, thus enhancing password strength (Li et al., 2017).

Li et al. were also able to study the difference between male and female users as the dataset included names and ID numbers (2017). The main differences they found were that male users generally seemed more inclined towards including personal information in their passwords, especially the user’s name and that their selection was typically less diverse, suggesting that the passwords were more vulnerable to cracking attacks (ibid.).
3. Method

The purpose of this chapter is to give a brief description of the method used in the thesis, how it helped realize the project and provide it with a sense of direction. The first section introduces the method while the second section elaborates a bit more on its application and the steps taken in the thesis.

3.1 Method selection

The purpose of the thesis was to design a working solution that should aid in solving the identified problem to the best and most effective extent possible. The expected outcome was a code-implementation of the artefact and thus, the obvious choice of method was one based on design research. Through the construction and evaluation of an artefact as a solution to an identified problem, research is addressed by design science (Hevner, March, Park, & Ram, 2004; Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). With this view, the research is approached in an iterative fashion where continuous evaluation and subsequent refinement of the artefact will take place. The goal was to try and improve practice by delivering something tangible and concrete that could be used to enhance the security in password-based authentication systems, and ultimately, the information stored in those systems.

The seven guidelines introduced by Hevner et al. (2004) provided the work with clear guidance to important aspects that needed to be covered in order to achieve research rigour, credibility and completeness:

- Design as an artefact
- Problem relevance – 1.2 Problem and purpose
- Design evaluation – 5 Analysis
- Research contributions – 1.5 Contributions
- Research rigour
- Design as a search process
- Communication of research

However, not all of them were addressed as they did not seem relevant, which incidentally is quite in accordance with the authors’ opinions who advocate that the guidelines be used based on the judgement of the researcher (Hevner et al., 2004). The guidelines are merely introduced here and are discussed in more detail in chapter 6, Discussion unless indicated otherwise above. The guidelines that coincide with the design research methodology introduced next are discussed in 3.2 Method realization.

While the above guidelines proved valuable throughout the work, perhaps even more important was the design science research methodology (DSRM) as proposed by Peffers, et al. (2007), that really provided the work with a roadmap. It bears a clear resemblance with the guidelines where a number of them are incorporated into a design science process, encompassing six steps:

- Problem identification and motivation
Leveraging an Active Directory for the generation of honeywords

- Definition of the objectives for a solution
- Design and development
- Demonstration
- Evaluation
- Communication

As suggested by the authors, the process need not be carried out in sequence nor does it always have to start with step one (Peffers, et al., 2007). Here, however, it is carried out in sequence as the research has been initiated from a problem-centred perspective, which according to the model has the first step as the entry point.

Moreover, it is believed that if this process is followed, in combination with some of the above guidelines, there is a good chance that research rigour will be achieved.

3.2 Method realization

The six above steps are addressed in the following section, some are described in more detail whereas others are merely pointing towards other sections of the work where they are being discussed more at length. Note that in the subsection design and development, a brief description of the process of software development is provided along with how it was applied in the development of the artefact.

I. Problem identification and motivation

The problem is identified and its motivation for research is given in the Introduction section as well as in section 1.2, Problem and purpose.

II. Definition of the objectives for a solution

The proposed solution and what will make it a successful one is touched upon in section 1.5 Contributions. The currently available solutions are discussed in the literature chapter, section 2.2 Existing algorithms for the generation of honeywords, and their respective efficacy in section 2.4, Algorithm comparison and evaluation.

Furthermore, while the suggested approach to solving the problem takes a slightly different standpoint than previous studies, the idea was not to detect flaws or deficiencies in algorithms used in those studies per se, but rather to make use of what had already been developed and apply it to the particular conditions and constraints of the present study. This line of thinking permeated the work as there is little to be gained from reinventing the wheel and wasting valuable resources. Whenever and wherever possible, using established techniques that serve its purpose well and could be applied in this context were chosen, meaning that the time available could be utilized in the most efficient manner.

III. Design and development

To be able to resolve the identified problem through the development of an artefact, at a conceptual as well as at a physical level, it was first necessary to establish a solid
Leveraging an Active Directory for the generation of honeywords

theoretical foundation. By ensuring a thorough investigation of existing theories and current solutions, steps towards achieving rigour were taken by: leveraging existing knowledge; benefiting from advantages of present solutions; and avoiding known limitations. In order to achieve an all-encompassing theoretical foundation, a comprehensive literature study was undertaken which is described in the following paragraphs.

Here, the first step was to survey the field so as to create a picture of what had been done up until this point and how it had been done. An examination of the existing algorithms used in creating honeywords was conducted in order to find one that was suitable either for direct use or as it turned out, to be used as a starting point for developing a new one that helped satisfy the goals. This part of the work was done by performing a literature review of what had been produced up until this point, comparing the results and evaluating benefits and drawbacks of different solutions and proposals. The results from the review served as input into the theoretical and conceptual construction of the artefact.

The next step was to investigate the anatomy of what plausible passwords look like by again consulting relevant literature. This was important so that the intended use of the information found in a typical AD could be leveraged to good effect. Of particular interest was the study of passwords that contained personal information, as this revealed user behaviour that could be leveraged in the construction of honeywords based on AD information. The compiled information constitutes a subsection of the literature review chapter and was important in order to establish a solid theoretical foundation for password usage that was necessary for the subsequent algorithm development.

The literature search was conducted as follows; the library publication databases at LTU as well as Google scholar were used to search for relevant literature. Keywords used when searching included honeyword, honeyword generation, passwords, password security, password cracking targeted password attacks, password construction, user password selection, leaked passwords, password protection, password vulnerabilities, personal information in passwords, PII, design science research. From the papers that were of interest, suggestions for additional papers to study could be inferred from works cited by the respective authors, as found in the bibliographies of the papers. The papers were then categorized according to their topic for easier handling. In addition, they were imported to Zotero, a bibliography tool that aided in the citation process for the thesis.

Finally, when a theoretical framework for how the artefact would be constructed and under what premises were in place, actual development could begin. The implementation of the algorithm was done using Java code, in the form of an application. As is typical of modern software development processes, an iterative and incremental approach was employed so as to be able to focus on continuous improvement and continued delivery. The process is described briefly below.
SOFTWARE DEVELOPMENT

The Software Development Life Cycle (SDLC) as described by Stephens contains six steps as depicted in the below figure (2015):

![Software Development Life Cycle](image)

*Figure 2 – The Software Development Life Cycle (Stephens, 2015, p. 277)*

The process starts with the gathering of requirements for the project. Requirements can typically be divided into functional and non-functional requirements where the former signifies what the system should do, how it should behave and what type of functionalities that should be included whereas the latter for example include quality, usability and security aspects (Sommerville, 2011). The main focus in this thesis has been on the functional requirements as the expected contribution is to resolve a known problem taking a novel approach by proposing a new conceptual solution, not to implement it in the best manner possible. The requirements were elicited based on the findings from the literature review and are described in the sections 4.1 Conceptual design and 4.1.2.1 Algorithm requirements. Additional constraints are introduced and discussed in 4.1.2.3 Algorithm adaptation.

The second phase of Design/Modelling seeks to present a design model that will be used as direct input of the subsequent phase of implementation. The architectural model shows the main structural components and their relationships and can in many instances be seen as a bridge between the first two phases of the SDLC (Sommerville, 2011). At times, the preferred output is two models, a high-level and a low-level one, where the former is more abstract so as to provide an overview and not distract with too many details, whereas the latter is detailed enough so that engineers can use it for subsequent implementation (ibid.). Here these two models are embodied in class diagrams that both show the different parts, i.e. classes and how they are related to one another. A general diagram providing overview is found in Figure 9 – Class diagram for PII-Syntax while the more detailed one displaying methods and variables is to be found in the Appendix.
The task of transforming the models into program code is the subject of the third phase, implementation, where the deliverable is a piece of working software. In an agile setting as is the norm today, the process consists of a series of iterations where every iteration produces a more capable piece of software, referred to as incremental delivery. The implementation process here consisted of three iterations that each added to the artefact’s capabilities, ensuring that all the requirements were met.

In the fourth phase, commissioners of the product may be able to test it to see how it conforms to the requirements and whether or not their expectations have been met to a satisfying degree (Stephens, 2015). As there are no external stakeholders in this project, all the testing was done continuously during the development phase, as part of the implementation.

An application’s deployment may need careful planning if it is to reside in a complex environment. The plan should try to anticipate every problem scenario as well as its resolution and if considered necessary, may also include a rollback plan (Stephens, 2015). The deployment process is the purpose of the fifth phase. For this thesis, since the application would be deployed in a testing environment, there was no need to design a deployment plan that had to be followed. The artefact was merely deployed in the same environment in which it had been built.

In the final phase, the deployed software is maintained, which for example includes bug-fixes, adapting to changing requirements, improving performance and the addition of new functionality (Pressman, 2010). Naturally, this phase is outside the scope of this thesis and was thus disregarded.

IV. Demonstration

A testing environment was set up where the solution was deployed and tested. Note that its purpose was merely to set up a platform on which the artefact could be deployed, not to evaluate the artefact itself. The idea was solely to try and reproduce a real-world application scenario. The evaluation of the artefact was not based on its performance in the demonstration part, but on the outcome, as discussed below. For the sake of convenience, a virtual environment was used. The different entities were:

- An AD server with representative data – represented by data in a database*
- A honeychecker
- An application server
- A client computer – simulating logons

*The data structure in an Active Directory is hierarchical in nature and makes use of the concepts of trees and forests\textsuperscript{10}. The many possibilities available and different

\textsuperscript{10} A tree is essentially a subdomain that is branching off a root domain where that subdomains together constitute a tree. A forest is a collection of several root domains and can be regarded as a single instance of the AD. Furthermore, both are container objects that can hold different organizational units. At the bottom of the hierarchy are the leaves that are the most basic components and is characterized by the fact that they cannot contain any child objects. For more information, see for example: https://technet.microsoft.com/pt-pt/library/ce759073(v=ws.10).aspx
application scenarios can make its application and administration quite complex. Thus, for the sake of simplicity, here the AD was represented by a flat relational database, as the sole purpose was to leverage the information found in the AD, not to present an accurate representation of how it is logically structured and physically stored.

The setup is depicted below in Figure 3.

The scenario is as follows: a client requests resources from the application server. Normally, the user is granted or denied access to resources based on the information in the AD for a particular user, i.e. user permissions. With the use of honeywords, the AD stores a list of sweetwords associated with each account. If the supplied password is in the list, it is supplied to the honeychecker who verifies whether it is a honeyword or the correct password. The honeychecker then either grants access or raises an alarm as discussed before. It should be noted that the honeychecker does not store any actual hashes of the sweetwords, they are all stored in the AD. Instead, the honeychecker only stores indexes of the sweetwords and evaluates all login attempts against these indexes.

The function of the honeychecker is to test whether a supplied password is a honeyword or not by matching it to a list containing all the honeywords created. In practice, this machine needs to be properly secured using standard hardening techniques so as to withstand potential attacks. As noted by Juels & Rivest, this form of distributed security means that should the honeychecker be compromised, the security is only reduced to the previous state of system authentication without the use of honeywords (2013).

Regarding the deployment of the actual artefact for the creation of honeywords, it was deployed on a separate machine where the creation of a list of honeywords for each user account was effectuated. This list was eventually stored on the AD server so that it would contain both the users’ passwords as well as the corresponding honeywords. Note that the list was not hashed for reasons already mentioned.

Finally, it may be argued that the demonstration phase may be more suitably described as a test phase as the main purpose in this context was to prove that the artefact would
work as intended as opposed to providing data that could later be evaluated and analysed for e.g. effectiveness and correctness.

V. Evaluation

Lastly, to be able to evaluate and analyse the findings from the study, suitable metrics were employed that were taken from previous studies when deemed applicable along with others, considered to be of relevance. Relevant metrics are outlined below and discussed in more detail in section 2.3, Metrics for evaluation.

- Flatness
- DoS resistance
- Multiple system vulnerability
- Usability impact:
  - System interference
  - Memorability stress
  - Typo safety
- Storage cost

The metrics were first used to evaluate and compare the existing generation methods so that a suitable algorithm could be found, to be used as a starting point for developing a new one. The metrics themselves were evaluated and scrutinized in order to determine which of them was more important for the particular setting of the thesis. As will become evident from the Analysis chapter, not all of the metrics were relevant.

After a suitable existing algorithm had been chosen, its evaluation would in a way serve as a baseline against which the proposed algorithm was later compared and evaluated.

VI. Communication

As for communication, it is presumed that the present report will in a way function as a means of communicating the results to those of which they may be of interest.

3.3 Alternate methods

As the goal of the work is to design an algorithm for the creation of honeywords, a design methodology would appear the obvious choice. However, different methods could well be used and are discussed in the following.

The principal outcome from a design science research project is typically an artefact of some sort (Hevner et al., 2004; Peffers et al., 2007; Venable, Baskerville, & Pries-Heje, 2016), that seeks to provide a resolution to a previously identified problem. What distinguishes design research from action research or action design research is the added dimension of a practical context in the latter methodologies. This context is most often embodied by an organization or corporation, where the proposed resolution to the problem is tested throughout its design in the context where it later should be deployed (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011). Action design research (ADR) as proposed by the aforementioned authors, actively seeks to address a problem
found in an organizational context through intervention and evaluation before ultimately constructing and evaluating an artefact (ibid.). Thus, had the proposed work been undertaken in an organizational setting, trying to solve a “real” problem, action design research had most likely been a better and more natural candidate for a research methodology.

Despite potential advantages that an organizational setting in which to deploy the artefact might bring, the added value of the outcome from this is in the author’s opinion limited. First, although genuine user data would be used as opposed to fictitious, evaluating the algorithm’s strength would most probably not lead to any novel insights that would not have been discovered in an experimental setting anyway. Second, and more importantly, in order to prove valuable, a list of passwords has to be compromised and then an intrusion attempt needs to be conducted by an adversary, where one of the honeywords is employed, thus leading to detection. This implies quite paradoxically that in order to demonstrate security, security needs to be compromised, i.e. to prove the algorithm’s effectiveness, a password list first has to become stolen, which naturally should be of the highest priority to prevent. From this it follows that it would not have been feasible to use an ADR method as this would have necessitated compromising organizational security.

A possible scenario, however, where ADR could have been used for the development and testing of the artefact would have been one where for example a limited amount of company data that are non-sensitive would be utilized, e.g. in a sandbox environment. A white-hat hacker\(^{11}\) could act as an adversary, trying to crack the password list and gain access to the secure network.

In conclusion, for the purpose of this study, it was neither possible nor deemed feasible to use ADR as a research method and hence a design science method was adopted as discussed above.

\(^{11}\) A white-hat hacker is a person who has the authorization to hack a system in an attempt to discover any weaknesses, vulnerabilities and security flaws that may exist so that these can be mitigated in order to strengthen the security of an organization.
4. Design and implementation

The chapter starts with an explanation of the design theories and concepts that constitute the foundation for the subsequent development of the artefact and the rationale for selecting the individual components. Next, the algorithm’s logical design is discussed before the chapter is concluded with a description of its implementation.

4.1 Conceptual design

Before implementation of the algorithm can begin, its design needs to be established at a conceptual level. First, the personal attributes used to make up the passwords are chosen. Then, requirements for a suitable honeyword generation algorithm are established. Next, based on the requirements, an algorithm is selected, adapted and customized.

4.1.1 Designing passwords from PII

Of the user information that is available in the Active Directory, the following attributes will be used to base the formation of passwords on.

- First name
- Last name
- Birthdate
- Account name

The choice of the attributes is in accordance with the findings from section 2.5.3 where these were found to be the most frequently used. Not included are the email address and phone number. Email address is not considered as it in a corporate environment is typically based on the formula firstname.lastname@company.com, thus not contributing any additional personal information. Phone number is neither included due to its low usage ranking. With regard to names, the only names considered here are the user’s first and last name. As concluded by Brown et al, friends or relatives’ names as well as nicknames can all be utilized in the construction of passwords (2004). Of these other categories of names, the only ones likely to be stored by an organization are names of families or relatives but these are for the sake of simplicity not included here. It would, however, be quite simple to include these should it be deemed relevant and the data be available.

The account name is equivalent to the username that is employed by a user to log in to the system and is typically assigned by the system administrator based on viable organizational authentication policies (Rhodes-Ousley, 2013). Typical constraints on the account name are; (1) a maximum of eight characters, (2) first character of the user’s first name concatenated with the user’s last name, (3) a number appended to (2), perhaps with a leading hyphen. The fictitious account names that are used in this thesis

---

12 These are just some examples of username composition constraints, different companies and organizations may employ different ones according to its needs and applicable policies. Its point here is not to be exhaustive but merely to exemplify its use and to provide data to be utilized in the thesis.
are on the form $First name_{l=1}, Last Name_{l=5}, Birthday_{l=2=YY}$ where $l = length$ and $Birthday$ are on the form YYMMDD. A sample table of the attributes is provided below in Table 6 where the rules for account name also are illustrated. The last entry in row number four is an example of an account name on the form used in the thesis.

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Birthdate</th>
<th>Account name</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>19760510</td>
<td>jsmith</td>
<td>2</td>
</tr>
<tr>
<td>Marty</td>
<td>Friedman</td>
<td>820615</td>
<td>mfriedma</td>
<td>1,2</td>
</tr>
<tr>
<td>Melanie</td>
<td>Stevenson</td>
<td>20011210</td>
<td>mstev-16</td>
<td>1,2,3</td>
</tr>
<tr>
<td>Vinnie</td>
<td>Moore</td>
<td>680130</td>
<td>vmoore12</td>
<td>1,2,3</td>
</tr>
</tbody>
</table>

*Table 6 – PII attributes*

As previously discussed, sweetwords are the total population of generated honeywords and the original password (sugarword), as illustrated below in Figure 4. The honeywords may or may not be based on the actual password.

For the purpose of this study, a new concept of *shamwords* is introduced. It holds the same relationship with sweetwords and honeywords and it is substituted for the sugarword as illustrated below in Figure 5 (next page). A shamword is thus a fake password upon which subsequently generated honeywords are based. It is associated with a specific user or user account and can in this respect be regarded as a honeyword itself, but it serves a different purpose and is thus named differently to be able to differentiate between the two.
In the report, the underlying idea is to first generate a shamword based on how users typically construct passwords leveraging personal information. Then, honeywords will be generated using an appropriate algorithm. Whereas the honeywords generated from a sugarword can but does not have to be based on the sugarword, in contrast, when using shamwords instead of sugarwords, the honeywords are always based on the shamword.

In keeping with the concept of combo-meaningful data as introduced by Shen et al., and concluded to be the by far most frequently used when attempting to categorize password usage (2016), the shamwords will be constructed from at least two different PII attributes. In effect, this means that a shamword could be composed of a name and a birthdate or of an account name and the user’s name or of all three attributes. Naturally, the shamword could also include other parts that would not be considered personal to the user, e.g. birthdate and a dictionary word or username and some random characters. This is illustrated in Figure 6 below: A shamword could either be composed of two or more attributes pertaining to group 1 alone or by choosing attributes from both groups, i.e. two from group 1 complemented by one or two from group 2.
4.1.2 Algorithm selection and adaptation

After the requirements have been established, from the previously introduced algorithms, the best match is decided on. This algorithm is then further refined and adapted as necessary.

4.1.2.1 Algorithm requirements

The honeyword generation algorithm must accept as input the shamwords generated previously and perform the desired password transformations subsequently. As discussed above, the shamwords are composed of different parts or categories of characters, each with a particular meaning and function. The groups are then joined together to form the shamwords. In order for a honeyword algorithm to take full advantage of the information and leverage it to good effect, it would be desirable that the groupings remain unaltered and the syntactical divisions intact. Moreover, it is necessary to be able to distinguish the PII and treat these attributes independently, e.g. a regular dictionary word may be substituted for another whereas a name belonging to the PII group may not; a random number may similarly be replaced by another while a date may require a different treatment.

Furthermore, the honeyword algorithm should be perfectly flat, i.e. from a distribution of sweetwords, the sugarword should not be more probable to distinguish than the honeywords. For the remaining security standards, resistance to DoS attacks and multiple system vulnerability is preferable. As for the usability standards, these are discussed next.

As was described in the theory section, honeyword algorithms fall into either the legacy or the modified-U1 category. The initial idea was to use an algorithm that was from the former category so as not to impact the usability negatively. However, since no user is actually going to be using the passwords as they will all be employed solely as a defensive measure against adversaries, this need will not be relevant and can thus be disregarded. In fact, none of the usability standards of system interference, memorability stress and typo safety will be applicable for the very same reason. This is depicted in Table 7 (next page) with N/A (not applicable). In addition, this implies that the usability standards will not be relevant for the evaluation of the constructed artefact either.

4.1.2.2 Algorithm selection

From the requirements stated above, it follows that the different chaffing methods are unsuitable as they do not consider groups of characters but treat them separately. However, chaffing-with-a-password-model might be applicable in the event that a dictionary word is included with the personal attributes in the formation of shamwords. Although Caps-key is also ignorant of character groups, however, since the transformation process is non-destructive in the sense that it does not alter the actual groupings but only individual characters within the groups, it may be a potential candidate. The same holds true for CNF that can be employed for instance in the transformation of dates.
Leveraging an Active Directory for the generation of honeywords

Algorithms based on the addition of a tail, i.e. take-a-tail, modified-tail and PDP are neither considered appropriate since the idea is really to add the tail to a real password before generating the honeywords. This process is a bit superfluous as the shamwords will be adequately prepared in its creation phase, adhering to applicable constraints, without the need for an additional phase of adding a tail. Likewise inappropriate are storage-index that only utilizes actual passwords and tough nuts which are not based on any information at all but only random characters. The hybrid method as described by Juels & Rivest contain chaffing-by-tweaking digits (2013) and is thus judged to be unsuitable for the reasons given above. At a conceptual level, however, a hybrid method is nothing more than a combination of serval methods and may thus be a realistic option should the individual methods be viable.

Of the eleven methods introduced in chapter 2.2, the final method not yet discussed is the modelling-syntax. As the underlying idea of this method is to use the same syntax for the passwords and the honeywords, it would appear to satisfy the requirements of not altering the syntactical groupings. Moreover, its performance using the metrics for evaluation introduced earlier is good in all areas except for the multiple system vulnerability. Table 7 below shows the score together with CNF and Caps-key.

<table>
<thead>
<tr>
<th>Honeyword method</th>
<th>System interference</th>
<th>Memorability stress</th>
<th>Typo-safe</th>
<th>DoS-safe</th>
<th>MSV-safe</th>
<th>Flatness</th>
<th>Extra storage cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling-syntax</td>
<td>N/A (No)</td>
<td>N/A (Low)</td>
<td>N/A (Yes)</td>
<td>High</td>
<td>Low</td>
<td>$\frac{1}{k}$</td>
<td>$k - 1$</td>
</tr>
<tr>
<td>CNF</td>
<td>N/A (No)</td>
<td>N/A (Low)</td>
<td>N/A (No)</td>
<td>Low</td>
<td>High</td>
<td>$\frac{1}{k}$</td>
<td>$k - 1$</td>
</tr>
<tr>
<td>Caps-key</td>
<td>N/A (Yes)</td>
<td>N/A (Low)</td>
<td>N/A (No)</td>
<td>Low</td>
<td>Moderate</td>
<td>$\frac{1}{k}$</td>
<td>$k - 1$</td>
</tr>
</tbody>
</table>

*Table 7 – Performance evaluation of modelling-syntax, CNF and Caps-key*

The conclusion from this section is thus that after having reviewed the existing honeyword generation methods, the decision is to use the modelling-syntax, slightly “embellished” with CNF and Caps-key.

4.1.2.3 Algorithm adaptation

Originally a three-step process consisting of the steps, tokenization, validation and generation, the process is reduced to include only two steps as the second one of validation is omitted. The function of this step is to validate the outcome from the previous step and flag any words that are not dictionary words but words that may be user-specific, i.e. PII. The user is then presented with the flagged words and given an opportunity to change them. As the proposed model will not rely on human intervention and the idea is not to exclude PII but rather the opposite, this step becomes redundant and is as a consequence omitted.

In keeping with the terminology introduced by the authors (Bojinov et al., 2010), the act of tokenization refers to the different character groups being converted into tokens,
Leveraging an Active Directory for the generation of honeywords

Each representing a different syntactical element. The difference here is that the process is reversed, i.e. instead of parsing a password into tokens, the tokens that are to constitute the shamwords are decided on first before being subsequently filled with characters. The process will be elaborated on and exemplified in the coming paragraphs. Moreover, the number of tokens from the original method has increased from mere **words**, **digits** and **special characters** to include the four PII attributes first name, last name, account name and birthdate. The available tokens are summarized in the below table; divisible means that it is possible to split the token, e.g. 7605 is divided into two parts with for example a dictionary word in between: 76apple05 but is still considered a single token; abbreviate refers to whether it is permissible to abbreviate a word or not, e.g. friedman becomes friedm or john simply j.

<table>
<thead>
<tr>
<th>Token name</th>
<th>PII</th>
<th>Divisible</th>
<th>Abbreviate</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>First name</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>PF</td>
</tr>
<tr>
<td>Last name</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>PL</td>
</tr>
<tr>
<td>Account name</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>PA</td>
</tr>
<tr>
<td>Birthdate</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>PB</td>
</tr>
<tr>
<td>Word</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>W</td>
</tr>
<tr>
<td>Digit</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>D</td>
</tr>
<tr>
<td>Special character</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
<td>S</td>
</tr>
</tbody>
</table>

*Table 8 – Token attributes*

Having introduced the available tokens, the next step is to make use of them as a means to construct the shamwords. This first step will be referred to as the **preparation** stage and replaces the original one of tokenization although the two are closely related. As mentioned above, it is basically the same with the difference that it is done in reverse since the intended outcome is a shamword as opposed to an analysis of an original password. The process is illustrated below in Figure 7 where (1) is the original three-step model and (2) is the modified version. The two steps are elaborated on in the following paragraphs.

*Figure 7 – Modelling-syntax process model*
The output from the two steps thus becomes:

<table>
<thead>
<tr>
<th>Model outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation phase</td>
</tr>
<tr>
<td>Generation phase</td>
</tr>
</tbody>
</table>

*Table 9 – Process model outcome*

The first phase of preparation aims to construct a shamword from information found in the AD. In order to be able to determine the number of possible variants, henceforth referred to as *shamword formula* or simply *formula*, some construction constraints need to be established. The constraints dictate the token usage and are for the purpose of this thesis as follows:

- PII tokens – minimum of 2
- PII tokens – maximum of 3
- Total tokens – minimum of 2
- Total tokens – maximum of 4
- PII tokens – limited to a single occurrence/formula
- Non-PII tokens – multiple occurrences/formula permissible

Since the focus is on utilizing personal information for the construction of shamwords, a minimum of two tokens from this category is required. To avoid overly complex formulas, the total number of tokens that may be used is limited to four. In addition, as already discussed in the literature section, do passwords rarely demonstrate such a complexity and thus the limit seems reasonable and in accordance with previous observations. Within a particular formula, a PII token from the same group cannot be used twice, e.g. it is not permissible to use the *first name* two times, regardless of whether the total number of tokens is two or four. However, non-PII tokens from the same group can be used twice, provided that the content is not the same, e.g. a formula of two dictionary words is permissible provided that the words are not equal.

From the given constraints it can be inferred that: if the formula consists of two tokens, they need both to be from the PII group; if the formula is made up of four tokens, there needs to be at least one that is not from the PII group; the only possible occasion when two non-PII tokens from the same group can be used is when the total number of tokens is four and the number of PII tokens is two. The number of possible permutations that exist for each token setup is shown in Table 10. The grand total when combined equals 1,008. This means that given the above constraints, it is still possible to generate shamwords in over 1,000 different ways, something that should be more than sufficient under most circumstances and ensure that a great variety is possible.

---

13 If desirable, other constraints than the ones presented here could be chosen, and the proposed model be updated and modified accordingly. Reasons may include demands for increased or decreased variation of the shamwords or to alter PII usage.
among the shamwords. It should be mentioned, however, that this is a theoretical number that is somewhat reduced in practice as further constraints regarding token positioning apply, constraints that will be introduced and discussed later.

![Table 10 – Number of formula permutations](image)

Another set of constraints concerns the length of the tokens. All tokens need to be of minimum length \( l \geq 1 \), while the maximum length is dependent on the combined length of all participating tokens as well as the number of tokens of the formula. For the special character \( S \) the length \( l \) is exactly 1, \( S_{l=1} \). For Words, \( W \), the length for the purpose of this work is restricted to be between 3 and 5 characters, \( W_{l=3} \leq W \leq W_{l=5} \). The reason is that here the focus is on utilizing PII in the making of shamwords and thus non-PII tokens should not make up too much space of the formulas and the shamwords. How the token length is calculated will be described in detail in the coming sections.

As for the positioning of the individual tokens in a given formula, the process is not completely random but some constraints apply here as well. This will also be discussed and clarified in the following sections that deal with implementation.

The second and final step is the generation of the actual honeywords. This step has two things in common with the original model: the output, which is honeywords; and that the generation of honeywords is based on the modification of character groupings, i.e. tokens. However, the treatment of the tokens is quite different as will be described in the following paragraphs where an example is provided to help describe the two phases of the process.

In the preparation phase, formulas are assembled from different tokens. The formula types, as well as its contents, are input to the generation phase. The decision how to treat the token is entirely dependent on its type. A brief example will help illustrate the process. A formula consisting of three tokens where two are PII might be: \( PF, D, PL \). The information for the PII tokens is retrieved from the AD and the digits are selected randomly. Here, the two names from the AD are Marty Smith and the digits 571. As is stated above, it is permissible to abbreviate a \( PF \) and Marty is here abbreviated to
contain only the initial 'M'. In this example, the tokens thus become: \( PF = M, D = 571 \) and \( PL = Smith \) and can be described on the formula \( PF_1, D_3, PL_5 \). The output from the first phase is thus the shamword \( M571Smith \) and the formula \( PF_1, D_3, PL_5 \).

The process continues with the second phase of generation which takes the shamword and its corresponding formula as input to begin the process of generating honeywords. Tokens are always modified separately in order to preserve their original function and each token is subject to different alteration methods depending on its type. The different methods available will be described shortly but are for the purpose of the present example as follows: the random digits \( D_3 \) are replaced by some other random digits; the \( PF_1 \) remains unchanged; and the \( PL_5 \) is either modified with the caps-key algorithm or left unaltered. Example honeywords may be as follows, the shamword in bold:

\[ M571Smith \quad M429Smith \quad M309smiTh \quad M681SmitH \quad M407Smith \quad M182smith \]

The available token modification methods are displayed in Table 11 below.

<table>
<thead>
<tr>
<th>Token name</th>
<th>Modification method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abbreviation</td>
</tr>
<tr>
<td>First name</td>
<td>X</td>
</tr>
<tr>
<td>Last name</td>
<td>X</td>
</tr>
<tr>
<td>Account name</td>
<td>X</td>
</tr>
<tr>
<td>Birthdate</td>
<td>X</td>
</tr>
<tr>
<td>Word</td>
<td></td>
</tr>
<tr>
<td>Digit</td>
<td></td>
</tr>
<tr>
<td>Special character</td>
<td>X</td>
</tr>
</tbody>
</table>

*Words are substituted randomly from an available dictionary*

Some key points that can be discerned are: PII tokens are treated more gently, i.e. less destructive alterations are applied, methods in common are abbreviation and no modification; non-PII tokens, on the other hand, are subject to heavier alteration where random substitution is a common method. Note that in a strict sense, the abbreviation method cannot really be regarded as a honeyword modification method, rather it is more of a token attribute that states whether or not it is permissible to abbreviate a word or date. Nevertheless, it is included here to demonstrate that it for the PII tokens is a valid option in the process of constructing shamwords and subsequent honeywords. Moreover, the process of abbreviation can be regarded to be
a form of modification. The probability for one method to be chosen over another as well as the rationale is explained and discussed in section 4.3 that covers the algorithm’s implementation.

Finally, an overview of PII-Syntax is provided below in Figure 8 that seeks to illustrate how the model and all of the constituent parts work together. The two main steps of preparation and generation, the respective outputs of formulas, shamwords and honeywords and how they all participate in the process can be discerned in the figure. At the bottom, a sample output is given.

![Figure 8 – Holistic overview of PII-Syntax](image)

To summarize, this section explained the conceptual design of the artefact, established the terminology and introduced the new concepts of shamwords and formula. The rationale for an algorithm was established before a suitable one was selected and then subsequently adapted in order to suit the introduced requirements and constraints. In the next section, the artefact’s logical design will be discussed.

### 4.2 Logical design

The intention of this section is to demonstrate how the logical design will be. This design will function as a blueprint for the artefact implementation that will follow. As the language chosen for implementation is an object-oriented one, the logical design
will show the participating classes and their relationships in a standard UML\textsuperscript{14} class diagram.

![Class diagram for PII-Syntax](image)

*Figure 9 – Class diagram for PII-Syntax*

The class diagram in Figure 9 can be regarded as a top-level diagram, meaning that it shows only the most important classes and their relationships. Not included are any utilities classes nor are any methods or variables. Its purpose is solely to provide an overview of the participating classes and how they relate to each other as well as how they communicate with external entities. A complete class diagram is provided in the Appendix.

4.2.1 Formula construction

The construction of the formulas is completely random, i.e. there is an equal chance for any token to be chosen to participate in the making of a particular formula. The same holds true for the formula length and the number of PII tokens. The ordering, or arrangement, of the individual tokens, however, is not completely random but is decided from the following constraints:

\textsuperscript{14} UML (Unified Modelling Language) is an object-oriented language that is typically used for modelling all kinds of systems and serves to aid in its visualization, thus facilitating system comprehension and construction.
4.2.1.1 Ordering

If the formula contains token \( T_{1,2} \), its position will be \( I_{1,2} \) with a probability of \( P \% \). Not every token is subjected to these constraints as is evident from the table below.

<table>
<thead>
<tr>
<th>Token</th>
<th>Position</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>PF, PL</td>
<td>1, 2</td>
<td>60</td>
</tr>
<tr>
<td>PF, PL AND (S OR D)</td>
<td>1, 3</td>
<td>60</td>
</tr>
<tr>
<td>PA</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>S</td>
<td>Last</td>
<td>75</td>
</tr>
<tr>
<td>D, D</td>
<td>2, 4</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 12 – Token positioning

The table should be interpreted as follows: if the formula consists of a \( PF \) and \( PL \) token, there is a probability of 60% that their positions will be 1 and 2; if there is a \( PF \) but no \( PL \) token, there is a probability of 80% that it will be positioned first in the formula; and if a special character \( S \) has been selected, there is a 75% probability that it will occupy the last position. In the event that 2 digits have been selected, they will occupy positions 2 and 4, the reason being that should they occupy 2 consecutive positions it would not be possible to determine where the first one ends and the second one begins as they eventually are populated with random numbers of random length.

The rationale for the constraints summarized in Table 12 is partly based on the findings by Li et al. (2017) and partly on assumptions of the author, suggesting that alternative constraints may be equally valid.

4.2.1.2 Length

The length of the individual tokens is determined in 2 steps, according to the following constraints:

In step 1, the \textit{formula length}, \( F_t \), i.e. number of tokens, determines the sum, \( T_t \), of the tokens’ length constituting a particular formula. For a formula length of 2 tokens, the total length of the participating tokens will be between 6 and 8; a formula length of 3 tokens will have a total length of between 8 and 10; and a formula length of 4 tokens will have a total length of between 10 and 12.
If preferable, it is possible to set a fixed value regardless of the number of tokens that makes up the formula. This is useful in circumstances where a password length policy is in place that dictates that passwords be of length \( n \), where \( n \) is a fixed value, e.g. 8. Further, for whatever reason, should other token lengths be desirable than the ones introduced above, it is a trivial matter to update the algorithm so as to conform to the required needs.

In step 2, the length of the individual tokens is set. The first token whose length is determined is the word \( W \), that can be of length \( W_{l=3...5} \). Its length is then reduced from \( T_i \): \( T_i - W_{l=3...5} \) to calculate the remainder. If there are any special characters in the formula, their length is already known as it is always \( S_{l=1} \). The maximum length of a token is dependent on \( F_l \) and \( T_l \) and can be calculated using the following formula, here with \( PA \) as an example: \( PA_{\text{max}} = T_i - F_i + 1 \). Finally, the length of the remaining token or tokens is then calculated from the remainder of \( T_i \) where all tokens’ length is selected randomly apart from the last one who will receive the value of the remainder to complete the equation. Two examples will help illuminate the process.

In the first example, the formula is \( PF, D, PL, PA \) and \( T_{i=12} \). \( PF_{\text{max}} = T_{12} - F_5 + 1 = 9 \). A random number between 1 and 9 is selected for \( PF \), e.g. 5. \( D_{\text{max}} \) then becomes \( T_{12} - 5 - F_4 - 1 + 1 = 5 \). The number 3 is selected for \( D \). \( PL_{\text{max}} \) is calculated in the same manner after which it receives 2, \( PL_2 \). The length of the final token \( PA \) is received from the remainder of \( T_i \) which is now \( T_{12-5-3-2}=2 \). The complete formula thus reads: \( PF_5, D_3, PL_2, PA_2 \).

For the second example, the formula is \( PA, PB, W, S \) and \( T_{i=11} \). \( W \) is determined first and becomes \( W_{l=4} \). In addition, we know that \( S_{l=1} \). The next token to be decided is actually the first one, \( PA \). \( PA_{\text{max}} = T_{11-4-1} - F_2 + 1 = 5 \). \( PA \) receives the maximum number of 5 resulting in the remainder of \( T_{i=1} \) which then dictates the only possible value for \( PB \) as 1, leading to the formula \( PA_5, PB_1, W_4, S_1 \).

4.3 Implementation, algorithm construction

The actual implementation of the algorithm followed the standard manner of construction in an iterative an incremental fashion as described in the method chapter. Three major iterations were distinguished with three corresponding increments:

- Iteration 1: Formula
- Iteration 2: Shamword – populating formulas from databases
- Iteration 3: Honeywords – constructing honeywords from the shamwords

Note that to implement the first phase of the process model from Figure 7, the preparation phase, the implementation is divided into two iterations while the second phase of generation is handled in the final iteration. The three iterations are briefly described in the following paragraphs.
Leveraging an Active Directory for the generation of honeywords

Iteration 1

In the first iteration, the goal was to set up the fundamental parts of the artefact, i.e. to construct the Formula, Token and the rest of the subclasses. This would enable the generation of formulas and tokens that adhered to the requirements and constraints as introduced earlier. A testing class was also constructed so as to be able to test the implementation continuously.

As the design requires that a number of decisions are made, some completely random and others with varying degrees of probability, a central function was to use a random number generator\(^\text{15}\) to help accomplish this task. The function is found in the Java class Random of package java.util and was implemented in the class Utilities (not shown in the diagram in Figure 9). In all instances where a probability rate is necessary, e.g. the token ordering as shown in Table 12, a random number is generated and depending on the range of the generated number, a decision can be made. For example, a probability of 75\%, is achieved if the random number \(R\) is \(1 \leq R \leq 75\) where \(R_{\min}=1 R_{\max}=100\). For values of \(R > 75\), the result is conversely false.

Iteration 2

In the second iteration, the next step was to expand the artefact from the previous iteration to include functionality for generating shamwords. The class Shamword was added which includes methods for populating a formula’s tokens with data. All PII-tokens and Word-tokens take their input data from databases, the Active Directory and Dictionary databases respectively. The two databases were thus also created in this iteration (recall that a database was used to simulate an Active Directory). For the sake of simplicity, both databases contain only the bare minimum of tables and columns. The Active Directory database consists of a single table employee that has 5 columns: id, first_name, last_name, account_name and birthdate. The Dictionary database has 3 tables: three, four and five which simply holds words of the corresponding length. The 2 columns are id and word. The id column in both databases is simply a unique identifier for each record which in database terminology is referred to as a primary key. It is of type integer and is updated using the common auto-increment function\(^\text{16}\). Next, both databases were populated with data, the Active Directory with fictitious employees and the Dictionary with English words of the appropriate length\(^\text{17}\).

The process of selecting a special character was done based on the findings by Shen et al. who have listed the 32 most commonly used special characters and their respective occurrence rate (2016). This list served as input to the random selection process so that

\(^{15}\) In computer science, it is very common to use an algorithm for random number generation that is deterministic, meaning that its output actually cannot be regarded as statistically random, however, provided that the algorithm is of sufficient standard, the result will pass many randomness tests and in most cases suffice (Stallings & Brown, 2015). The generated numbers are referred to as pseudorandom.

\(^{16}\) Auto-increment is a database feature of the DDL language that allows a unique number to be generated automatically whenever a new record is inserted into a table without having to specify it explicitly. It is commonly used for generating primary keys.

\(^{17}\) The Dictionary database was populated with 500 popular English words for each column which were retrieved from http://www.thefreedictionary.com
the probability of selecting a particular character would match that of actual real-world password usage as investigated by the authors.

The digit tokens $D$ were populated using the random number generator discussed previously by simply passing the length as an argument. The only constraint is that all tokens of length $l > 1$ must not start with a ‘0’, i.e. $D_{l=2} \geq 10$. For $D_{l=1}$ $0 \leq D \leq 9$.

**Iteration 3**

In the final iteration, support for the generation of honeywords from the previously constructed shamwords was added.

The Caps-key method was implemented that changes from between 1 up to 3 minuscule characters per token to majuscules. The number of alterations $N$ is again random with the following probability distribution: $N_1, P = 60\%$, $N_2, P = 25\%$ and $N_3, P = 15\%$. The inclination towards the lower range of alterations is due to the fact that as established in section 2.5.2 on password usage, users are typically quite lazy when constructing passwords and thus are likely to minimize the use of capital letters and furthermore, because many password policies state that only a single uppercase character be used.

Recall that the idea of the CNF method is to alter digits in a manner so as not to alter their original meaning, e.g. if the digits resemble a date, the output of the method should still be a valid and plausible date. The implementation was as follows: first, a decision whether to subtract or add to the original value is made with an equal probability for either one. Next, a value of between 1 and 3 is either subtracted from or added to the original value based on the decision from the previous step. Finally, the new value is checked to see if it still a valid date. If not the process is repeated with a change of sign from the first step, i.e. addition becomes subtraction or vice versa. Note that this check is only performed on the $PB$ token as this by definition will always be a date. When the CNF method is used on the $D$ token, the check is redundant.

Random substitution of a token means that the population process is simply repeated, i.e. the token is repopulated anew with different data following the exact same process as the initial one. Note that the attributes of the token remain unchanged, it is only the content that is substituted, e.g. a Word token of length 4, $W_4$ will still be $W_4$ after the substitution but its content *bean* has been replaced by a new word *pear*. Also, note that a new value for the random number $R$ is calculated that may influence certain decisions. These principles apply to all three instances when the random substitution method is available, i.e. for all of the Non-PII tokens Word, Special character and Digit.

**Honeyword method selection**

Every token has at least two different honeyword alteration methods from which to choose, including the *no modification* as one and disregarding the abbreviation one as previously discussed. The probability for a method to be used in the honeyword construction process over another is illustrated below in Table 13. The rationale for the different values is discussed next.
For the PII tokens $PF$, $PL$ and $PA$ there is an evenly distributed probability to use the Caps-key method or to do nothing at all. As previously discussed, PII tokens should on the one hand only be subjected to moderate alteration whereas on the other hand, there needs to be some modification taking place or else no honeywords can be generated. This is of course directly related to how the formula is comprised. For a formula consisting of e.g. two or three of the PII tokens above and no Non-PII token, there is a greater likelihood that the Caps-key method needs to come into play. Had the formula contained e.g. a Word, it may have sufficed to do most of the modification there by substituting it for other words and leaving the PII tokens unaltered.

For the Birthdate token $PB$, there is a 25% probability of the CNF to be used and a 75% probability that no alteration is done. The reason for the relatively low probability rate to use CNF is that the whole adversary scenario assumes that someone with knowledge of the organization’s employees and some of their PII may be able to dismiss the altered records as not being genuine, thereby decreasing the effectiveness of the honeywords.

As for the Non-PII tokens, their methods in common are random substitution and no modification. For the Word token $W$, there is a 50% chance of choosing either method. This is so as to achieve a balance between immediately substituting a word’s content for another and allowing some words to remain unchanged and thus participate in more than a single honeyword. The Digit $D$ is the only token to have three methods to choose from. It is an 80% chance that some modification to a Digit token will take place, divided equally between the random substitution and CNF methods. The two methods are merely used for achieving variation and balancing the distribution. Finally, for the Special character $S$, there is a probability of 75% that it will be substituted for another special character. For the $S$ and $D$ tokens, the reason for performing no modification is the same as for the Word.
Tools

Finally, the software tools and programs that were used for the implementation are summarized. The tools were chosen at the discretion of the author and needless to say, other tools could have been used that would have yielded the same results.

- Eclipse – Oxygen
- MySQL – Workbench 8.0
- VirtualBox – Oracle VM 5.2.16

Eclipse is an integrated development environment, IDE, that among other things can be used to program applications, primarily in Java. MySQL Workbench is a visual database design software that handles the administration and development of a relational database management system (RDBMS). VirtualBox enables a virtual environment to be set up on a host computer. All of the mentioned programs are open-source.
5. Analysis

In this chapter, the algorithm is evaluated using the metrics as introduced in chapter 3 and discussed in 2.3, Metrics for evaluation. First, some examples of the output from the algorithm are provided.

5.1 PII-Syntax sample output

In this subsection, four examples of output from the algorithm are given that serve to demonstrate the results in the form of shamwords and honeywords.

Example 1

In the first example, the formula consists of 3 tokens which all belong to the PII category: $PF_2, PB_3, PA_4$. The entries in the AD for this example are displayed below:

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Birthdate</th>
<th>Account name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vinnie</td>
<td>Moore</td>
<td>761110</td>
<td>vmoore76</td>
</tr>
</tbody>
</table>

The shamword consists of the tokens $PF_2 = Vi, PB_3 = 761, PA_4 = vmoo$ that combine to make up the complete shamword: ‘Vi761vmoo’. When generating 20 honeywords, the sweetword population $Sw$ of length $l$ totals $Sw_{21}$. An example output is shown below, the shamword in bold:

Vi761vmoo Vi761vmoo VI761vmoo VI751vMoo VI761VMoo VI751vmoO
Vi761vmoO Vi750vmoo Vi751vmoo VI761VMOo VI761VmoO VI761vMOo
Vi761VmoO Vi761vmOo VI761VmoO VI760vmoo VI761VmOo VI761vmoO
Vi760vmoo Vi761vmOo Vi761vMoo

Things to note from the honeywords generation include; the $PF$ token is altered in exactly half of the 20 cases; the $PB$ token remains unaltered in 14 of the 20 cases, or 70%; and the $PA$ token is altered in 14 cases, or 70%. Both the $PF$ and $PA$ tokens use the Caps-key honeyword method whereas $PB$ uses the Close-number-formation CNF. For the 14 cases of the $PA$ token, the number of characters altered is: 1 character: 9 instances, or 64.3%; 2 characters: 4 instances, or 28.6% and 3 characters: 1 instance, or 7.1%. Also note that $PB$ is in the interval $750 \leq PB \leq 761$, even though the CNF is limited to $-5 \leq PB \leq +5$. The reason is that a number such as ‘759’ which by definition is in the range of the CNF method, $756 \leq PB \leq 766$ in this example, does not yield a valid date. For the date form of YYMMDD used in this thesis, the only valid number for the third position is ‘0’ or ‘1’. The result is that the only valid number in the range for the CNF method other than the original ‘761’ is ‘760’. As a consequence, it is necessary to alter the number in the second position to allow for more possibilities according to the formula $-2 \leq PB_{pos=2} \leq +2$. This in combination with the different
possible values for the third position yields a total of 9 different combinations, which is only 1 less than the normal CNF output of 10.

**Example 2**

For the second example, the formula is of length 4 with 3 PII tokens and 1 Non-PII token. The formula $PA_2, PL_5, PB_4, S$ gives rise to the shamword ‘jsSteph9805!’ as it is populated with the below data.

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Birthdate</th>
<th>Account name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jennifer</td>
<td>Stephenson</td>
<td>980505</td>
<td>jsteph98</td>
</tr>
</tbody>
</table>

For $Sw_{21}$ a sample output is as follows, the shamword in bold:

<table>
<thead>
<tr>
<th>jsSteph9805!</th>
<th>jsSteph9805#</th>
<th>JsSteph9805!</th>
<th>JsStEph9805.</th>
<th>jstEph9805&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>jSStEph9804!</td>
<td>JsStePH9805;</td>
<td>jsSteph9805&amp;</td>
<td>jSteph9805:</td>
<td>JSSteph9805*</td>
</tr>
<tr>
<td>jSSteph9805@</td>
<td>jSStePh9802/</td>
<td>jsSteph9805*</td>
<td>jStepH9803-</td>
<td>jSteph9805&amp;</td>
</tr>
<tr>
<td>jSSteph9807.</td>
<td>jSteph9803.</td>
<td>jsSteph9805$</td>
<td>jSteph9802!</td>
<td>jSStEph9805!</td>
</tr>
</tbody>
</table>

Things to note are; available methods for the tokens $PA$ and $PL$ are Caps-key, for $PB$ CNF and $S$ random substitution. $PA$ is altered 9 times, or 45%, $PL$ is altered 7 times, or 35%, for $PB$ the number is 6 times and 30% and for $S$ the numbers are 15 times and 75% respectively. In the 7 instances where $PL$ is altered, the number of characters modified is; 1 character: 5 times, or 71.4%; 2 characters: 2 times and 28.6%. The $S$ token is substituted for 10 different characters where ‘.’ and ‘&’ are most common with 3 occurrences each. $PB$ is in the range of $9802 \leq PB \leq 9807$. As $PB_{pos}=4$, there is no need to apply the same logic as in the previous example with the only exception that ‘9800’ is a non-legal number since $01 \leq PB_{pos=3,4} \leq 12$.

**Example 3**

In example three, the token length is again 4, this time, however, the token distribution is even between PII and Non-PII, i.e. 2 each. The formula $PB_1, PL_2, D_4, W_3$ is populated with the below data from the AD to make up the shamword of ‘8Sm8703bat’.

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Birthdate</th>
<th>Account name</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>881012</td>
<td>jsmith88</td>
</tr>
</tbody>
</table>

$Sw_{21}$ shows the 20 generated honeywords below:
Things to note here include: honeyword methods used for the PII tokens are CNF and Caps-key, for $W$ random substitution is used and for $D$ CNF or random substitution is employed. The single-character $PA$ is altered only twice, or 10%, $PL$ is altered 5 times, or 25%, $D$ is altered 15 times, or 75% and $W$ is substituted for another word 10 times, or 50%. Of the 15 times that $D$ is modified, CNF is used 5 times resulting in its occurrence ratio equalling that of no modification at 25% out of the total cases while the random substitution is used in 50% of the honeywords. None of the substituted words $W$ occur more than once.

**Example 4**

The fourth and final example consists of a formula with the minimum length of 2, $PL_4, PB_4$. From the below AD information, the shamword ‘Rhoa6811’ is formed.

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Birthdate</th>
<th>Account name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randy</td>
<td>Rhoads</td>
<td>681105</td>
<td>rrhoa68</td>
</tr>
</tbody>
</table>

The sample 20 honeywords are shown below:

<table>
<thead>
<tr>
<th>Rhoa6811</th>
<th>RHOa6811</th>
<th>RhoA6811</th>
<th>Rhoa6811</th>
<th>RhoA6810</th>
</tr>
</thead>
<tbody>
<tr>
<td>RhOA6811</td>
<td>RHOA6811</td>
<td>Rhoa6809</td>
<td>Rhoa6808</td>
<td>RhoA6810</td>
</tr>
<tr>
<td>Rhoa6809</td>
<td>RhoA6810</td>
<td>Rhoa6809</td>
<td>Rhoa6811</td>
<td>Rhoa6808</td>
</tr>
<tr>
<td>RhoA6809</td>
<td>RhoA6809</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Things to note are: for the $PL$ and $PB$ tokens, the methods used are Caps-key and CNF respectively. $PL$ is modified 16 times, or 80% and $PB$ is modified 12 times, or 60%. Of the total of 16 times, $PL$ is altered to have 1 extra majuscule 10 times, or 62.5%, 2 extra 5 times, or 31.3% and 3 extra a single time, or 6.2%. The range of the output from $PB$ is $6808 \leq PB \leq 6813$. Finally, it can be mentioned that in order to produce the desired output of 20 honeywords, the algorithm had to run 165 times to avoid any duplicates. The reason, of course, being that the fewer the tokens in a formula the greater the chance that either the no modification method is proposed or that the alterations made coincide with such already made.
5.2 Evaluation

In this subsection, the PII-Syntax algorithm is evaluated by metric.

5.2.1 Flatness

Recall that flatness is one of the most important metrics in the evaluation of a honeyword generation algorithm as it in essence determines its effectiveness, if an algorithm lacks sufficient flatness, it can be potentially easy to discern the real password from a population of sweetwords. Typically, an algorithm’s flatness is an indication of the relationship between generated honeywords and the real password, in this case, it is the relationship between honeywords and the shamword, which in effect is also a form of honeyword. This raises some questions as to its relevance here as a valid and important metric. These concerns are being further discussed in the next chapter. Nevertheless, its inclusion here serves at least two purposes; to ascertain a comprehensive artefact evaluation and not least to be able to evaluate the proposed model against previous methods.

All of the honeyword generation algorithms that are part of the PII-Syntax, i.e. Caps-key, Close-number-formation and modelling-syntax all show excellent flatness individually. From a population of sweetwords $S_w$, the probability of guessing the correct password is $\frac{1}{k}$ where $k$ is the number of elements in $S_w$. For the random substitution to acquire the same flatness value, the honeywords need to be distributed like real user passwords from the perspective of the adversary, which is quite in accordance with the tweaking methods as discussed by Juels & Rivest (2013). The fact that they are always selected randomly, be them words, digits or special characters, ensures that this criterion is being met. It follows then that if all parts that make up PII-Syntax are perfectly flat, at least under the conditions given above, then PII-Syntax itself must also be perfectly flat.

5.2.2 DoS resistance

The security against Denial of Service attacks is strongly related to the overall security policies of the system and the password policy in particular. A strict password policy may stipulate that upon the submission of a single honeyword, all accounts are locked and a global password reset policy is issued or that the system is put in a lockdown state. A more liberal policy, on the other hand, may be to only lock down a single account at a time. In the former case can an adversary in possession of a password list willingly submit a single honeyword causing the entire system to be shut down. Or an employee may do the same by accident. Whatever policy is chosen is of course dependent on a number of things including for example sensitivity of the system, the likelihood of being attacked and other existing security measures in place. It is normally the recommendation that a medium policy is applied that is strong enough to offer an acceptable security but not so strong so as to increase the susceptibility of being hit by DoS attacks.

Of the honeyword methods that are used in the PII-Syntax, CNF and Caps-key offer weak resistance to DoS attacks whereas the modelling-syntax is not regarded to be very
susceptible to this type of attack. According to Juels & Rivest, the hybrid model and the chaffing-with-a-password-model are safe against DoS attacks (2013). From this, it can be inferred that random substitution of tokens other than words, i.e. special characters and digits are safe as well. Moreover, since the PII-Syntax is a hybrid model in that it employs and combines a number of different ways to create honeywords, it is considered unlikely that an adversary be able to guess any existing honeywords and use these to mount a DoS attack.

The attack scenario above presupposes that a user is in possession of a password as discussed by Chakraborty & Mondal, Erguler and Juels & Rivest (2017; 2016; 2013) and can successfully guess honeywords based on the real password, either with or without knowledge of the honeyword generation method used. In the case of the PII-Syntax model, the shamwords used for the generation of honeywords should not be known to anyone but perhaps the system administrator or security staff. Thus, the scenario where an employee may be able to successfully guess honeywords based on knowledge of the underlying password becomes impossible.

To summarize, it is argued that the PII-Syntax model offers good resistance to DoS attacks and the most probable attack scenario is not applicable by the very nature of its construction.

5.2.3 Multiple system vulnerability

The property of multiple system vulnerability or MSV is the resistance of an algorithm to the intersection operation. An adversary that has obtained password lists from two different systems can perform the operation on the two distinct lists to find a match for the same user, thus retrieving the real password.

As with any other honeyword generation algorithm, the PII-Syntax produces different outputs each time it is run, i.e. the same shamword will give rise to different sets of honeywords for every time it is run. Should the same shamword be used for several systems, it suffers the same vulnerability against MSV as most honeywords algorithms do. However, since the shamword is an automatically generated word that is not intended to be used as a password by a real user, it is recommended that new shamwords be created when a user needs to appear in several passwords lists for authentication in different systems. Thus, if all shamwords are unique, the resistance to MSV will be high. Nevertheless, since the shamword in itself is also a fake password that when used will trigger an alarm, refraining from changing it when used in multiple systems will in effect cause an adversary who performs the intersection operation on two compromised password lists to identify it as occurring twice and thus drawing the conclusion that he has encountered the real password. This turns the weakness of being susceptible to the MSV into an advantage, as the attacker is misled to draw false conclusions.

The conclusion is thus that either way the PII-Syntax is used in multiple systems, i.e. with a single unique shamword or several different ones, the resistance to MSV is sufficiently high. This is of course if the question of multiple systems is even relevant to begin with.
5.2.4 Storage cost

The cost of storing the honeywords is completely dependent on the number of honeywords per user account that is used. The effectiveness of employing a honeyword scheme for authentication, however, is also directly dependent on the number of honeywords used. The more honeywords used, the greater the chance that an adversary will choose it, leading to subsequent detection. Between these rivaling interests, i.e. algorithm effectiveness vs minimum storage, a balance needs to be struck. Juels & Rivest propose that a number of around 20 honeywords per sugarword is sufficient in most circumstances to achieve the desired level of security without increasing storage requirements too much (2013).

For a system with \( n \) users where the number of honeywords for each user account is represented by \( k \), the total amount of sweetwords can be calculated either with the formula \( n(k - 1) \) or simply \( nk \), depending on whether the honeywords are based on a sugarword or a shamword. In this thesis, the latter equation is thus used and for a system of 1 000 users and \( k = 20 \), the resulting number of sweetwords that needs to be stored is 20 000. It follows that for any number of \( k \), the storage size required increases by \( k \) times.

Assuming that all sweetwords are stored in a hashed format with for example the SHA-512 algorithm would mean that every sweetword be of size 512 bits which is equivalent to 64 bytes. For an account where \( k = 20 \) as above, the total storage space required would be 1.25 kilobytes (kB). A system that has 1 000 user accounts would thus require 1.25 kB x 1 000 = 1 250 kB ≈ 1.22 Megabytes (MB). If we were to include the salt into the equation using a 128-bit value, for the system described with a thousand users, the sum would add up to 312.5 kB. Thus the total storage space required for all 20 000 sweetwords and salts would still be less than 2 MB. Moreover, it should be mentioned that a password hash of 512 bits and a salt of 128 bits are considered to be very large, reducing the size to 128 and 64 bits respectively would not affect security significantly.

From the above reasoning, even for systems with significantly more users, it is concluded that the cost of storing the additional honeywords is in most cases a negligible one. Furthermore, in light of the increased security honeywords can provide, the cost of a minimally increased storage is undoubtedly an acceptable one.

Besides the physical space needed to store the sweetwords, one needs also to take into account the extra processing power and memory required in the systems to handle the larger amounts of data. Being required to handle larger amounts of data could potentially lead to an increase in system latency, something that may impact usability as the system is perceived as slower. However, for the example system of 1 000 users, it is presumed that the impact is negligible at most.

In sum, the drawbacks of the increased storage required to store the honeywords are indubitably balanced by the enhanced security, and as for the PII-Syntax, it is on par with most previous algorithms in this regard.
5.2.5 Summary

Finally, the results are summarized and contrasted to the performance of other relevant algorithms as can be seen below in Table 14. In addition, it should be noted that since all the honeywords are generated automatically by the PII-Syntax and not intended to be used by any human, the usability standards of system interference, memorability stress and typo safety need not be considered in the evaluation of the artefact.

<table>
<thead>
<tr>
<th>Honeyword method</th>
<th>DoS-safe</th>
<th>MSV-safe</th>
<th>Flatness</th>
<th>Extra storage cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>PII-Syntax</em></td>
<td>High</td>
<td>High</td>
<td>$\frac{1}{k}^*$</td>
<td>$k - 1$</td>
</tr>
<tr>
<td>Modelling-syntax</td>
<td>High</td>
<td>Low</td>
<td>$\frac{1}{k}^*$</td>
<td>$k - 1$</td>
</tr>
<tr>
<td>CNF</td>
<td>Low</td>
<td>High</td>
<td>$\frac{1}{k}^*$</td>
<td>$k - 1$</td>
</tr>
<tr>
<td>Caps-key</td>
<td>Low</td>
<td>Moderate</td>
<td>$\frac{1}{k}$</td>
<td>$k - 1$</td>
</tr>
<tr>
<td>Tweaking**</td>
<td>Low</td>
<td>Low – High***</td>
<td>$\frac{1}{k}^*$</td>
<td>$k - 1$</td>
</tr>
</tbody>
</table>

*Table 14 – Comparison of PII-Syntax to other honeyword algorithms

* honeywords need to be distributed like real user passwords from the perspective of the adversary
** tweaking resembles random substitution
*** depending on the type of characters tweaked

As is evident from the table, the PII-Syntax performs equally well or better than all of the other algorithms. The only exception is that to be perfectly flat, some conditions apply.

To summarize, the results clearly indicate that PII-Syntax is capable of producing honeywords of good quality that adhere to the prevailing security standards in all regards. Based on a sound theoretical foundation, it capitalizes on existing solutions and advancements while seeking to add a new dimension through the incorporation of users’ personal information.
6. Discussion

This chapter starts with a discussion of the results from the previous chapter. Next, some reflections on the methodology used are given along with a brief discussion of the study’s limitations before the chapter is concluded with a discussion of several other topics and aspects that have surfaced during the writing of the thesis, some of which needed elaborating on.

6.1 Results discussion

One might argue as to the appropriateness of using the flatness parameter to measure the algorithm’s effectiveness since what it actually measures is in effect how much a honeyword stands out from the real password. For an algorithm to be considered completely flat, there should exist an equal probability that any of the words in a sweetword population are chosen. In the case of the PII-Syntax algorithm, measuring its flatness is really comparing the honeywords to the shamword, which by definition also is a fake password in its own right. Thus, in order to fulfil its purpose, it suffices if any of the words in a sweetword population derived from a shamword is chosen, including the shamword itself.

However, the impression is that ignoring to evaluate the artefact against this central metric would have left a sense of incompleteness to the analysis. Moreover, the fact that its inclusion makes it possible to compare it to previous algorithms and measure its performance against these should not be underestimated. After all, when trying to develop something new, it is always valuable to be able to compare it to previous efforts within a particular area.

In addition, as evident from the results, it demonstrates the artefact’s quality with regard to flatness, a fact that both helps to provide reassurance to potential stakeholders and to align it with existing theory.

It can be argued that the Multiple system vulnerability, or MSV property is not entirely applicable in the current context as the use of an AD as a means for authentication in an organizational setting typically permits the use of Single-Sign-On (SSO), which in effect eliminates the need for several password lists. Nevertheless, it is still not uncommon to find several different means of authentication in a large organization that has to cope with legacy systems running alongside new ones. Thus, the security property MSV is still viable. Should the need to have more than a single centralized authentication system exist, as noted earlier there are two possible ways to approach the honeywords generation process using the PII-Syntax. Reusing the same shamword in several systems would still lead to different honeywords being produced at runtime, at least in theory, meaning that a user would have different honeywords on the systems but that they are based on the same shamword, which of course also would be present on the systems. Although the generated honeywords would most likely be different there is no guarantee that any or all actually are. This is especially true in scenarios where the shamword formula does not allow for a large population of honeywords to be generated, e.g. if the formula length is only 2. The advantage of using the same shamword in several systems that could potentially lure an adversary into believing he
has found the real password will quickly be demolished if two of the honeywords are also identical. Thus, it is recommended that a new shamword be generated for every system that a user is registered with.

Even though the algorithm by itself shows strong resistance to DoS attacks, the opinion is that these types of attacks should preferably be stopped at the company perimeter. Instead of relying on the algorithm to be invulnerable to DoS attacks, whenever encountering numerous login attempts, they need to be thwarted at the server side before ever reaching the honeychecker.

Most of the existing honeyword methods incur storage overhead to some extent. This has been identified as one of the major weaknesses of the honeywords concept (Juels & Rivest, 2013). Efforts to remedy this has been undertaken by the research community, notably by Chakraborty & Mondal and Erguler (2016; 2016). Both of these studies, however, focus on the storage cost only in terms of how much more data that need to be stored physically. As demonstrated earlier, these numbers are not insignificant but are indeed very small, a few Megabytes for a system of around 1 000 users. While not a negligible aspect, the increased security that is provided by using a honeyword-based approach for authentication does indeed make the effort worthwhile. It is the author’s opinion that the focus instead of size requirements be on storage cost in terms of system overhead and processing times. Should the use of honeywords increase a system’s response times significantly, this would obviously be a major deficiency and a strong case for opting out the use of honeywords. Thus, this aspect definitely merits more attention and could well be a topic for continued research.

As for the experimental setup that was used to set up a working real-world application scenario, the outcome from this was no more than to offer a bit of assurance that the solution could well work in this context. In retrospect, this step might have been a bit superfluous as the initial intention was to perform more tests and analyses. System performance, in particular, is one of the things that could have been interesting to study more in depth, had time permitted.

As a final remark, it should be mentioned that the objective was never to outperform previous algorithms but rather to approach the problem from a different perspective by leveraging the information from an AD. That said, the proposed solution should obviously not perform worse than existing honeyword methods for it to be a viable alternative for any organization to adopt.

For the proposed artefact to be the prime candidate for honeyword generation in an organization that chooses to use the honeyword strategy as one part of their defence, the results clearly indicate that PII-Syntax performs on par with or better than existing algorithms when evaluated against the aforementioned metrics. In addition, the inclusion of users’ personal information for the construction of honeywords that is firmly rooted in the theoretical findings of how users tend to choose their passwords brings an added value of believability. Finally, as evident from the experimental setup which in essence can be regarded as demonstrating proof of concept, PII-Syntax is both simple to deploy and has the property of being easily adaptable to fit in most contexts and under most security policies.
6.2 Methodological reflections

Hevner et. al. affirm that the building and application of a design artefact lead to knowledge and understanding of both the design problem and its proposed solution, and for that reason, the artefact as such is essential in any design science research undertaking (2004). The artefact can be manifested in different ways, including for example a method, model or an instantiation (ibid.). The PII-Syntax presented in this thesis fits equally well with all three definitions: it is a method with a clear purpose that can be followed to meet posed requirements in a specific context; it is a model that shows how the artefact is designed and constructed; and its implementation is an instantiation of the artefact itself. Moreover, it is described in sufficient detail, both at a conceptual level as well as at a practical level with implementation in mind, so as to enable the evaluation and suitability in a particular application context.

The artefact’s rightful existence stems from the definition of a problem that is both of importance and whose resolution cannot be said to be completely satisfactory despite numerous attempts made by the research community. The risk of information theft or disclosure due to password breaches will continue to be a viable and imminent threat for as long as passwords will be used for system and account authentication. As one way of responding to the problem, the proposed solution pursues one fairly recent research direction while approaching it from a new perspective. It is the hope and belief that the inclusion of PII in the construction of honeywords will be beneficial to the overall security in password-based authentication systems.

As the requirements and constraints derived from the solution and the conceptual design have been satisfied, the artefact can be said to be both complete and effective (Hevner et al., 2004). Relevant metrics were utilized that were adopted from existing research and enabled the artefact’s comparison to previous models. The contributions from the construction of the artefact is a novel manner in which to approach a well-known problem which ultimately leads to an extension of the existing knowledge base. The capitalization of present knowledge and its application in a different way is another contributing factor that may help expand the knowledge in the area. Finally, the thorough investigation into existing theory and knowledge as evident from the literature review, in combination with an emphasis on artefact effectiveness, comprehensiveness and applicability, and clear research method utilization help assert research rigour.

6.3 Limitations

As the focus has been on password-generation based on information from an Active Directory, the solution will only be applicable in scenarios where an AD is used. This means that it will not be applicable for public websites and the like but instead will find its use in organizations and corporations which are prime users of an AD for authentication and authorization purposes.

For reasons discussed earlier, it was not possible to employ an Action Design Research, (ADR) method. However, had it been used, advantages such as organizational evaluation and adoption may well have contributed to the quality of the artefact. The absence of this may be viewed as a limitation of the study.
When studying the use of personal information in passwords, there is a very limited amount of material available. While there is a fair amount of leaked password files available from previous breaches, the information in these files is constrained to only include account information and the corresponding password. To be able to study if and how users leverage personal information when constructing passwords would require additional data about the account owner. In the present work, the section on PII usage is mostly based on a single paper by Li et al. (2017), resulting in a theoretical underpinning that may neither be very generalizable nor provide sufficient depth. As a consequence, there is a possibility of a skewed analysis due to the limited data from a single dataset. Nevertheless, their findings corroborate earlier results by for example Brown et al. and Castelluccia et al. (2004; 2013) and are consistent with other observations by for instance Shen et al. and Veras et al. (2016; 2012). Thus, it is the author’s opinion that there is a sufficient amount of information that does suffice in establishing a solid theoretical foundation.

The findings in the above paper by Li et al. also included gender differences in the password selection process (2017). This was something that was not included in the development of the algorithm, but that may have contributed even further to the quality of PII-Syntax.

Finally, the results of the study were not evaluated quantitatively as in the paper by D. Wang et al. (2017) due to resource constraints, but instead in the same heuristic manner as most previous reports, e.g. (Chakraborty & Mondal, 2016; Erguler, 2016; Juels & Rivest, 2013). Had the results been assessed in this manner and subjected to the same experiments, there is a possibility that the outcome would have been different.

### 6.4 Additional remarks

To avoid the risk of exposing the real passwords after having generated the honeywords based on the shamwords, it is necessary to generate some honeywords that are based on the actual passwords used by the system users. The reason for this is that if every user account has one sugarword and one shamword associated with it and honeywords are generated only based on the shamword, the result will be a group of related words and a single one, i.e. the sugarword. In this scenario, it is possible for an attacker to detect the group of sweetwords based on the shamword as being related and derived from one of the members, while the actual sugarword is quite possibly unrelated and thus easily distinguishable from the rest. This is illustrated in Figure 10 (next page). The natural mitigation to this vulnerability would be to generate additional honeywords that are based on the sugarword to disguise it, which of course would be the classical application of the honeywords concept. Critics may argue that this is a flaw in the algorithm when it rather depends on how it is applied. The algorithm is independent of the environment in which it is deployed and its only function is to construct shamwords based on a user’s personal information and then generate honeywords. However, as the overarching goal is to be able to detect password list breaches, it may be required to use the algorithm together with another one that will generate honeywords that are based on a user’s real password. While this scenario has not been tested as it is deemed to be outside of the scope of the thesis, a few suggestions as to how it may be implemented is discussed in the following paragraphs.
Depending on how the users’ passwords are constructed, the choice of honeyword method may vary. However, assuming that they are mostly made up of meaningful bits, as research bears evidence to, the suggestion is to use the modelling-syntax as this capitalizes on this very fact and because it offers a good level of security. The ratio between how many honeywords should be generated from which model does not necessarily have to be equal, i.e. for $S_w$ where $k = 20$, there is no need to have 10 honeywords generated from each model. Rather, it may prove beneficial to introduce some natural variation here as well. Figure 11 below and continued on the next page provides 2 examples to help illustrate the idea.

Naturally, using another honeyword generation algorithm together with the PII-Syntax will have consequences on the overall security. Should the other algorithm exhibit a significantly lower level of security, then it is by this level that the overall security has to be measured. Therefore, it is of utmost importance that the complementary model in all aspects be of good quality.
Questions as to the usefulness of deception techniques such as honeywords for thwarting attacks on password-based authentication systems have been raised, and this section addresses some of these. First of all, the scenario in which honeywords will make a difference is when an entire password file has been compromised and an adversary tries to leverage it to get unauthorized system access. This assumes that an offline attack has been launched, which for various reasons is not considered to be of the same probability as other online attacks at an attacker’s disposal, for example; malware, phishing and sniffing (Bonneau et al., 2015). As a consequence, the authors argue that the focus on making the passwords as hard to crack as possible has wrongly been given too much research merit, something which should have been put to better use elsewhere (ibid.).

However, simply dismissing it is a threat of low probability when both reality and literature are full of examples of the contrary borders on ignorance. Because information security is a multi-faceted subject, there will never exist a universal solution that will offer complete protection to every threat imaginable. Therefore, it is argued that deception techniques such as honeywords have earned themselves a rightful place as one piece of the security puzzle.

Recall that the principal attack scenario is when a password file has been compromised and deciphered, which might suggest that the scenario is somewhat unlikely to ever occur. However, there is ample evidence that password files have been and probably will continue to become stolen. Even though it for quite some time has been regarded a security best practice to hash and salt passwords, empirical observations indicate that more than 40% of sites do not store the passwords in a hashed format (Bonneau et al., 2015). And of those who do, the use of salt cannot be taken for granted as in for example the case of the LinkedIn and last.fm breaches of 2012 or the more recent VNG breach of 201518.

---

18 The website https://haveibeenpwned.com/PwnedWebsites is one of the websites where details about password breaches are available.
Targeted password guessing, where an attacker can scan social networks to obtain personal information is a great threat to all password-based authentication systems as it is quite common for users to use this kind of information when constructing passwords. All honeyword generation methods are vulnerable to this type of attack (Chakraborty, Singh, & Mondal, 2017). The method presented in this work flips the objective and takes advantage of this very fact; the personal information from the AD used to create the honeywords ensure that it is the honeywords rather than the real passwords that will use PII, thus increasing the likelihood that they will be chosen by an attacker.

Sometimes users choose password and account names between which there is some form of correlation. Correlational hazards can arise when there is a direct, discernible relationship between the username and password. For example, if a username such as “football” is chosen with the password “Messi” then there is a correlation between the two login credentials and the creation of honeywords cannot effectively disguise the original password (Chakraborty & Mondal, 2017) nor the correlation. All honeyword generation algorithms that construct honeywords based on a user’s real password are susceptible to this vulnerability. From this, it follows that PII-Syntax is not vulnerable to this risk as it does not use real passwords as input to the honeyword generation process. Furthermore, while perhaps not all that applicable to the current context as users rarely get to choose their account names in an organization, it is still worth mentioning as it is a potential risk.

Finally, it may be argued that some of the formulas used in the thesis for the construction of shamwords and subsequent honeywords are overly complex and may not adequately produce the desired illusion of a real password. There is, however, evidence to suggest that users are moving in the right direction and starting to adopt stronger passwords, (Shen et al., 2016) something that will help justify its use here. The authors also conclude that as a consequence of using more complicated passwords, there has been an increase in the use of combo-meaningful data including PII, (ibid.) and thus the tokens used here to make up the sham- and honeywords merely try to reflect that. Moreover, as the implemented algorithm is designed with easy adaptability in mind, reducing complexity is a trivial matter of changing some of the parameters, e.g. reducing formula and token length.
7. Conclusions

The purpose of this study was to investigate whether an Active Directory could be used as a resource in the construction of honeywords. The assumption was that since the AD contains information about real system users, it should be possible to leverage this information for the creation of high-quality honeywords because of the very fact that they are based on actual users.

In the introduction of the thesis, the rationale for the research problem and the proposed solution was discussed and the context in which they appear. In order to be able to draw on existing knowledge, a comprehensive literature study was undertaken with the purpose of establishing a solid theoretical foundation. The findings were used both to distinguish an appropriate model that could be customized in order to meet all requirements and to gain a thorough understanding of password usage. The knowledge acquired from existing theory was put to use in the development of the new model with the intention of capitalizing on previous progress as far as possible. From the established requirements, a conceptual model was developed first that would satisfy the requirements for PII usage in password construction and the subsequent honeyword generation. Next, a logical model was developed in the form of a class diagram that would function as a roadmap for the implementation of the algorithm. Finally, the algorithm was implemented.

The proposed model has been compared to existing honeyword generation models with regard to flatness, DoS resistivity, multiple system vulnerability and storage cost. The results show that PII-Syntax in all regards performs equally well or better than existing models. More importantly, however, is the fact that since it introduces personal information from actual system users to the equation, there is an enhanced level of believability with regard to the generated honeywords. Regardless of the type of honeyword algorithm used, at the heart of all of them is the fact that for the algorithm to be successful, believability and credibility of the output is key. Moreover, as is evident from studies on user password selection and construction, users are prone to base passwords on meaningful information which justifies its inclusion in the proposed solution.

In conclusion, using the information that resides in an Active Directory for the generation of shamwords and honeywords as a defence mechanism against password theft and system compromise will have positive effects on overall system security. The results clearly indicate that PII-Syntax performs equally well or better than existing honeyword generation algorithms and because of the fact that users show an inclination towards including personal information in their passwords, leveraging the information from an AD will enhance the quality of the generated honeywords.

Suggestions for future research

- Is the modelling-syntax honeyword method the best choice for usage in conjunction with PII-Syntax? Are there other alternatives that will complement it better or equally well?

To avoid exposing the sugarword when using PII-Syntax, additional honeywords should be generated that are based on the sugarword. The
modelling-syntax has been assumed in this work because of its close resemblance with PII-Syntax. Continued research and experimentation may lead to new knowledge from which informed decisions can be made whether superior alternatives exist that should be adopted.

- **Can the deployment of the PII-Syntax model in a real-world application scenario be leveraged for a more comprehensive performance evaluation, leading to a refined model?**
  
The use of the model in a real-world context with real data can lead to new insights, be them additional requirements, updated constraints or new conclusions, that in turn can necessitate changes to the model.

- **Will the effectiveness of PII-Syntax benefit from more human-controlled input?**
  
  Most of the decisions that are used in the model are made randomly, e.g. formula construction, selecting whether to use a honeyword method or not, and if so, which one. Based on a deeper understanding of password composition and continued testing, motivation for model refinement may be warranted.

- **What are the effects of an increased storage cost on system performance?**
  
  Using PII-Syntax or most other honeyword algorithms comes at the expense of an increased storage cost. So far, the literature has only been concerned with storage cost as in the extra physical space required to store the honeywords, not with how (if) it also affects performance.

**Concluding remarks**

With the ever-increasing reliance on computer systems for information storage, processing and retrieval, given the value of the information itself, securing the data processes becomes paramount. As one tool in the security toolbox, using deception techniques in general and honeywords in particular, it can help strengthen security in numerous instances as practically all systems use some form of authentication and a majority of them use passwords.

If PII-Syntax in any way can contribute to strengthening the security of password-based authentication systems, then the efforts undertaken here will have been worthwhile. It is the hope that the findings here can be put to use in real-world application scenarios as opposed to just purely academic contexts. Should that prove not to be the case, however, it will at the very least have extended the knowledge base through the exploration of a novel path.
8. References


Leveraging an Active Directory for the generation of honeywords


Leveraging an Active Directory for the generation of honeywords


Leveraging an Active Directory for the generation of honeywords

Management Information Systems, (3), 45.
https://doi.org/10.2753/MIS0742-1222240302


Leveraging an Active Directory for the generation of honeywords


Appendix

Class diagrams

A class diagram in more detail is provided where all the participating classes, as well as all methods and attributes, are included. Note that simple getter and setter methods have not been included to improve readability. Also, note that the classes outside of PII-Syntax are not classes in a strict sense, they have just been illustrated as such for the sake of convenience.

The diagram is divided into three pages, the first one (present page) provides an overview of the classes, their relationships and cardinalities. The remaining pages display the contents of the individual classes in more detail.

![Detailed class diagram – overview](image-url)
Leveraging an Active Directory for the generation of honeywords

Class diagram with attributes and methods – left part
Leveraging an Active Directory for the generation of honeywords

Class diagram with attributes and methods – right part