Master’s Programme (60 credits) in Network Forensics

E2PM: Enclosed Portable Password Manager

Aung Naing Oo
Passwords have been a necessary evil for a while. Today’s computer users have multiple accounts on the internet, with each burdening the user’s memory with complex long passwords. The requirement of generating, memorising and maintaining such passwords is becoming a bottleneck that modern-day password managers try to alleviate to a certain extent. This is all good until the stored password vaults are breached on the user’s machine or third-party servers where passwords are stored get infiltrated. Hence there is a need for something more obscure and self-contained. In this research, E2PM, which stands for Enclosed Portable Password Manager, is a hardware-based password manager that aims to be self-contained, secure, and portable. These three attributes are achieved by using a live operating system that fits on a portable flash drive whose contents are encrypted using the AES-256 algorithm. E2PM can be used through a live boot or the Virtual Box application. Passwords are stored in a separate partition on E2PM’s drive, never touch the host computer’s hard disk and are strongly encrypted. E2PM intends to provide a low-cost solution using existing hardware as compared to contemporary hardware-based password managers and provides backwards compatibility, which means the user needs not make any drastic changes to their application data.
ACKNOWLEDGEMENTS

I would like to give a big thank you to my professor Dr. Wojciech Mostowski for supervising me and giving me great feedback throughout this project. It has been a pleasure working under him.

I would also like to thank my examiners and Professors Eric Järpe and Mark Dougherty for taking their time to read my report.

Last but not least, I would like to thank Halmstad University for providing me with a platform to formalize my idea in the form of a thesis. This idea is something that I have always wanted to work on for the past few years. Hence it has been a pleasure to pursue it formally.
INTRODUCTION

In 1979, Bell labs published a paper on password security for the time-shared UNIX system. The idea is to grant access to computing resources only to those who are authorized to use them [15]. Users, of course, can run an open system without any password, but for systems that require a high degree of security, a password must be present. Passwords also provide another layer of security against unauthorized access to computing resources even if the user is logged in. For instance, the “root” password on Unix based system safeguards against the execution of code that can alter the system. Only a few authorized people may have access to that password, and it is not meant to be disclosed otherwise. Another safeguard passwords provide is against the unauthorized access by ex-employees who may not be under any direct control and may have intimate knowledge of the business and the surrounding systems.

Unintended disclosure of passwords can have a devastating effect on the organization. It is mentioned in the paper that during maintenance, a system administrator was editing the password file, and another system administrator was editing daily messages for printing to the connected terminals. The design flaw in the system led to an interchange between the two files and led to the printing of passwords in plain text on all connected terminals. This lapse resulted in high administrative cost since everyone’s passwords had to be changed simultaneously.

Traditionally, passwords on UNIX systems are stored in encrypted format under "/etc/passwd" [8] (now under "/etc/shadow"). The encrypted format is usually an output of a one-way hash function applied on a plaintext password input. Passwords for other modern-day applications are always stored in encrypted format in databases or files. Theoretically, any password can be cracked given unlimited time and computing resources [18]. However, it is improbable for an attacker to reverse a password hash given the finite amount of time and computing resources presently available on the market. Moreover, a salt string is also
usually appended to the password to make it harder to crack using methods such as Rainbow tables.

Despite that, simple passwords can be easily cracked using freely available tools such as John the Ripper, Cain and Abel and Ophcrack [18]. Brute-forcing is a common technique where the attacker generates all possible combinations of letters and repeatedly input the generated password into the one-way hash function. If the result produces the same output as the stored hash in the password file, it is a match, and the attacker succeeds in gaining access to the system. There are other techniques such as Rainbow tables, dictionary attacks and hybrid methods whose main idea is to reduce the amount of time it takes to guess the password since brute-forcing takes a very long time. Hence, long passwords are encouraged and enforced in cases to prevent such attacks.

In practical systems, usually, there are also additional safeguards such as throttling and account locking upon several consecutive unsuccessful password attempts. Also, the actual password is never stored, rather the hash of the password is stored. The user’s input of password is then hashed and compared against the stored hash to see if it is a match as shown in Figure 1. A salt, which is a randomly generated sequence of strings, is also usually stored and concatenated with the plain text password for verification. This helps improve the complexity and strength of the stored password.
1.1 PASSWORD MANAGERS

Figure 2: KeePass Master Password input interface. An alphanumerical key of the user is required before KeePass unlocks the credential vault.

Users are now faced with the challenge of not only generating but also remembering complex long passwords. Studies show that human beings are generally poor at retaining more than seven characters in memory. Some users then would reach out to the easier way of writing down their passwords in notebooks which is the ultimate irony because the very idea of putting a password is to prevent others from ever finding out one’s password. It is commonly known that human beings are the weakest link in the CIA (Confidentiality, Integrity, Availability) triad, and most of the famous breaches in organizations happen because of a weak link [17].

Figure 3: KeePass Interface. Once the master password is successfully authenticated, the password vault is unlocked and credentials in plain text can be retrieved.

This is where password managers come in to alleviate the situation. The core idea of a password manager is to allow the
user to store user-ids and passwords of various accounts in a safe vault. This vault is encrypted using symmetric encryption techniques such as AES-256, which require a secret key. This key then becomes a master password which the user must remember and key in every time the vault is to be accessed (decrypted and read from). The advantage is that, instead of a user remembering multiple passwords for different accounts, she would need to remember one master key for her vault. The disadvantage is that the vault itself is stored on the computer or on a third-party server which could be subjected to breaches by attackers.

The most popular form of password managers such as KeePass shown in Figures 2 and 3 come in software format, either open source or proprietary. Hardware-based password managers are a work in progress and there are only a limited number of solutions to choose from. More on this will be presented in the next section. Most modern-day software-based password managers can generate passwords with a very high entropy value and store them in a vault. But to prevent a breach of these stored vaults or to entrust a third party with critical information [7], existing measures are not sufficient, or there is a lack of study in this direction. The problem with hardware-based password managers is that generally, they are not backwards compatible, and adopting them usually requires big changes at the architectural level in an organization.

This research aims to bridge the gap between the two solutions. That is to significantly reduce the exposure of the password vault to unintended or malicious access either through the network or through a virus infection on the host machine. And to prevent the addition of a whole new layer to the existing hardware in addition to catering for backward compatibility.
1.2 Research Question(s), Problematization and Limitations

This entire research is undertaken by a need to put the user in control of their passwords, and also with the goal of the passwords never touching the host system’s hard disk to prevent leakages. Hence, the following research questions arise:

- Has this problem been addressed before? If so, what kind of solutions exist, and what are their drawbacks? (This is addressed in Chapter 2)

- If the goal is to prevent the passwords from ever touching the host system’s hard disk, how does one create such an isolated system? (This is addressed in Chapter 3)

- How does the proposed system compare to existing solutions? (This is addressed in Chapter 4)

To problematize the research questions into one compact sentence, there is no readily available password management solution on the market that is 1) portable, 2) cost effective and 3) backwards compatible, that gives the user full access and control of their passwords. Hence, this research is done to fill the gap.

In terms of the limitations, E2PM needs to be tested and surveyed on actual pool of users to get some statistics on usability. Hence, this data is currently not available. The device is also not supported yet on mobile phones, Chromebooks and Personal Digital Assistant devices such as Kindle readers. Due to limited time and scope, extensive penetration testing has not been done on the implementation.

1.3 High Level Proposal of E2PM

The principle usage of E2PM is such that the user plugs this device into their computer and accesses the custom password manager through the built-in Midori browser. The password manager and its supporting applications are included in the Core System, which is immutable, while the passwords are stored in the Secured Data partition. The user can use E2PM through Live Boot or through Virtual Box. The immutability
of the Core System ensures integrity of the password manager and supporting applications packaged into E2PM.

Figure 4: High level architecture. The blueprint of E2PM presented in three parts; Core, password manager and Secured data partitions.

Without going into too much technical detail in this section, a high-level architecture of the prototype is presented in Figure 4. The core system, which is a read-only partition, will be powered by a Linux kernel with a total image size of around 1 to 2 GB. The password manager program will be located as part of this read-only partition within the core system. The password manager is responsible for managing the secured data in the read/write partition. This is where the safe vault is stored. The entire system can fit into a USB drive of around 4 GB in size. A user would be able to run the password manager either through a live boot session from the USB or by interacting with the USB drive through the Virtual Box application.

The user would need to select E2PM drive on the BIOS screen to do a live boot. When E2PM gets loaded, it will start a small server listening on port 8080 to provide password management service. This service can be accessed through Midori browser which is pre-installed during the image creation process. The password management portal will store and fetch new records from the secured data partition only on E2PM’s partition. The usage of E2PM is essentially the same on Virtual Box except for a few differences during the set up stage. It will be presented in details in the later sections.
Normal use case

Figure 5: Use case flowchart. There are two ways in which one may use E2PM, either through VirtualBox or live boot from E2PM.

Figure 5 presents a flow chart that demonstrates how one would typically use E2PM. When an actor plugs in E2PM into the USB slot on their computer, they have the option to either use it through VirtualBox or through a live boot (as explained in the previous section) which requires a computer restart followed by selection of the right boot device from the BIOS menu.

If VirtualBox is to be used, the user does not need to reboot the computer.

Once E2PM is loaded, the user can access E2PM’s password manager through Midori browser by hitting the exposed URLs. The user should already have set their master key password of 16 characters in length. If the master key password is accepted, the user can either read their stored credentials or append new password entries to the list. A user would temporarily need to remember the password entry they see on the screen, since no copy/paste feature is enabled.
Once the user is done using, they can shutdown E2PM by issuing ‘shutdown’ command on the terminal, plug out E2PM from the USB, and restart the computer (if not to be used any longer). It is highly recommended that the user restarts the computer or shuts it down when no longer is use. This is to prevent any attack on the physical RAM as described in more details in Chapter 4.

Optionally, the users are more than welcome to use any of their old computers as a standalone device for E2PM usage only.

The following chapter will present existing research done on hardware and software-based password managers. This will then allow the reader to see further how the proposal fills the gap between hardware and software-based password solutions.
Currently, in the market, there exist two types of password managers in the form of both hardware and software. Purely software-based password managers are more popular and widely adopted by both individuals and organizations compared to hardware-based ones not just because of the cost savings but also due to convenience factors such as ease of deployment and usage. This paper has surveyed both the scientific and consumer communities on the various new technologies being developed to improve the secure storage and retrieval of passwords.

2.1 PASSWORD ENTROPY

One might wonder what type of password would be strong enough to protect against attacks. There are intuitive guidelines by security experts that one can follow to set such a password. But there is also a quantitative way of measuring the complexity of a password and that is through entropy measurement. In information theory, entropy is defined as a way of measuring the uncertainty of a random variable X which can have a finite set of values $x_1, x_2, ..., x_n$ with $P(X=x_i)=p_i$, where $0 \leq p_i \leq 1$ for each $1 \leq i \leq n$, and $\sum_{i=1}^{n} p_i = 1$

Mathematically, the entropy measurement of X is as follows:

$$H(X) = \sum_{i=1}^{n} p_i \times \log_2 \frac{1}{p_i}$$

where, by convention, $p_i \times \log_2 \frac{1}{p_i} = 0$ if $p_i = 0$.

And, $H(X) = \log_2(n)$ if and only if $p_i = \frac{1}{n}$ for each $i$, $1 \leq i \leq n$ (that is, all outcomes are equally likely) [13].

This means, for a simple password of five characters in length drawn from alphabets a to z, the size of n would be $26^5$. $p_i$ in this case would be $\frac{1}{26^5}$ since all outcomes are equally likely. Hence, the formula reduces to simply $\log_2(n)$. For the given example with 5 characters, $H(X)$ would be $\approx 23.5$. 


The higher the entropy, the harder it is to crack the password. It would require $2^{H(X)}$ attempts to guess the correct one. Entropy value can be increased by expanding the character space and the length of the password.

With a high entropy, it takes longer to crack a password given the finite computing resources of today. So, in most cases, it is safe to have a password length of at least eight characters with a combination of mixed case and alphanumeric variations. The challenge now is for the user to remember this password. There are also accessible tools like web applications that one may use to check the password strength. One such example is passwordmonster.com\(^1\).

\(^1\) https://www.passwordmonster.com
2.2 SOFTWARE BASED PASSWORD MANAGERS

The mechanism of a software-based password manager is to allow the user to create a master password which is then used to encrypt and store the countless number of logins and passwords for different applications. A modern-day password manager would also come with other functionalities, such as allowing the user to store secure notes, credit/debit card numbers and other information which the user would want to keep secured. Another common feature is to generate complex and strong passwords for the user that can be used to register and login into websites and other applications. Of course, such generated passwords would then have to be made accessible to users across multiple devices and different operating systems. Since human memories can only hold about seven characters, it becomes rather simple for modern-day computers to crack such simple passwords in a short period of time. Hence, a password manager comes in to alleviate the burden of remembering long and more complex strings.

According to an article from the Guardian, the author complains about how difficult it is to generate and remember a password such as "thera1n1nSpa1nstayzma1ny1nthedra1nSW1A2AA", which according to Microsoft is strong enough to be considered unbreakable by present-day computing power. However, it involves at least four steps and significant mental juggling to come up with one as such. Moreover, there is no assurance that the user would be able to recall it after some time of inactive usage has passed. The simplest form of a password manager would be to store passwords in an Excel spreadsheet and encrypt the file with a master password. But then, it becomes a problem when one wants to access this Excel file across different computers or on computers where programs to parse XLS format are not available.

2.3 DASHLANE, KEEPASS, LASTPASS AND 1PASSWORD

There have been studies on at least four popular password managers, namely, 1Password, LastPass, KeePass, and Dashlane. Apart from KeePass being open source, the other three are all proprietary, with only Dashlane providing a free plan for one user for up to 50 passwords on just one device. All three of them have subscription plans for individual, team and enter-
prise users. 1Password is known to be used by several media companies such as CNN, BBC, Mashable, and The New York Times. It has a social media presence and an extensive support page where most of the issues and FAQs are documented. There is also a Reddit page where users can visit and engage with the community. A unique feature of 1Password is the presence of Watchtower, which is meant to report to the user about password breaches [4]. It does that by checking against the information available on haveibeenpwned.com². In terms of the core functionalities offered, all three of them are similar. The only difference is in terms of online community presence and user support mechanisms such as ticket handling response time and 24x7 user support.

Another study performed a usability analysis of these four password managers by surveying 14 participants [2]. Five evaluation criteria were set to perform the tasks, namely:

1. Efficient: A measure of how fast a user can perform a task accurately.

2. Effectiveness: A measure of whether an intended task is completed successfully and with accuracy.

3. Engaging: Is the user interface pleasant and satisfactory for usage?

4. Easy to learn: Is the user interface easy to learn without deliberate effort?

5. Error tolerant: A measure of how well the software handles user errors.

There were five different tasks that the participants were asked to perform, which included 1) registration, 2) login, 3) remote login from another computer, 4) changing an existing password of a particular website, 5) logging in with the new password. The participants then rated the tasks on a 7-point Likert scale ranging from Totally Satisfied (TS, 6) to Totally Dissatisfied (TD, 0).

The research [2] mentions that Dashlane outperformed its three competitors on all the five evaluation criteria. From the technical perspective, the study also conducted a security analysis of all four password managers. The analysis was divided into three main security goals, namely:

² https://haveibeenpwned.com/
1. SM | Security of the master key
2. SDDB | Security of the credential database
3. SC | Security of communication exchanges across different devices

Once again, Dashlane ticks all the boxes to meet the security goals specified. For instance, Dashlane imposes a minimum mandatory length and complexity requirements for the master key, which is stored securely. It encrypts the credential database with AES-256 and gives feedback on the security levels of the passwords stored. It provides support for strong password generation on behalf of the user and allows for multifactor authentication. In terms of secure communications, it uses HTTPS with TLS v1.2, AES 128 and Ephemeral Diffie-Hellman Exchange to communicate with external servers. AES-256 is used for Dashlane’s communication between the browser plugin and the password manager.

The study also quantified the comparative analysis of the leading security goals. SC was excluded because the cloud storage functionality was not applicable to all the four password managers and is not a mandatory feature for a password manager to function. Dashlane scores an average overall security score of 1.8 while KeePass and LastPass are on level pegging at 1.33.
2.4 KEEPASS

KeePass\(^3\) is an open-source password manager that anyone can
download, build, modify, and use as he/she sees fit. The official
source code can be downloaded from Sourceforge, a website
for hosting open-source code bases. Issues related to bugs are
raised by independent users on SourceForge, and the commu-
nity tries to fix them depending on how critical the bug is. Since
the project is open, KeePass’ source code has been extensively
reviewed and evaluated. One such code review was done by
EU-FOSSA, which is an initiative by the EU to audit free and
open-source projects to qualify the project as fit to be used by
the institutions within the EU \[^3\]. This also benefits the public
since potential bugs and vulnerabilities can be discovered and
flagged for public awareness.

The code review focused mainly on the security aspects of
KeePass, namely: the risks that certain aspects of security pose
to the user and the confidentiality and integrity of the data
in the KeePass database. The KeePass database is known to
be encrypted using AES and Twofish algorithms. Overall, the
code review did not find any critical or high-risk vulnerabili-
ties, but there were five medium risks discovered which mainly
revolved around the concept of secure coding practices, such
as handling proper type conversion, allocating sufficient mem-
ory for an object, avoiding systems calls that could result in ex-
portable vulnerabilities and avoiding the usage of deprecated
functions such as rand(). Others were mostly low risks and of
informational categories.

KeePass also does not provide any cloud syncing function-
als and leaves it to the unofficial channels or independent
contributors to provide such extra functionalities. There are sev-
eral unofficial KeePass ports available for different operating
systems, web browsers and mobile devices.

2.5 OTHER NOVEL SOFTWARE-BASED SOLUTIONS

There have also been other research and experiments in cre-
ating novel solutions for software-based password managers.
One such example is that of Steganography based password
manager \[^9\]. Steganography is the technique of hiding data in
\[^3\] https://keepass.info/
files, images, sound files, and videos. This research argues that there are several issues related to password management systems, primarily that master keys can be attacked, information stored on a local machine can be leaked or attacked through freely available tools and the lack of trust in web-based versions of password management tools. The author then proposes a steganographic solution in which the password and login data for applications are encrypted and stored in a chosen media object, such as an image. The user then must input a master key and choose the correct stego object to retrieve the login credentials of an account or application. As novel as this idea is, the portability aspect of the password manager is completely ignored. There can be several instances where a user does not have access to the machine where the stego objects are stored and hence would not be able to access the login credentials.

Meanwhile, some studies are conducted to better understand what drives the users to adopt the idea of having software-based password managers. A user’s perception of password loss and its severity has a high impact on the adoption of a password manager in an organizational context, even though the user may not necessarily trust the product or the technology behind a password manager. Hence, there is a need to give users other alternatives that they can truly be comfortable with and have peace of mind knowing that their credentials are only available in one place. At the same time, all the good attributes such as ease of use, portability, accessibility, and security of software-based password managers are preserved.
2.6 Hardware Based Password Managers

Just like its counterpart, the goal of hardware-based password managers is to securely allow the user to store and retrieve passwords but with the added security feature of not storing the passwords on the computer’s hard disk to prevent leakage through cyberattacks and malware. Syncing of password databases to the servers of third-party software-based password managers also becomes unnecessary since the hardware can be easily carried around.

According to Stajano [20], a new method of authentication should have at least three main aspects fulfilled. They are as follows:

1. Memoryless: User need not memorize any secrets.

2. Scalable: There should be no drop in performance of the device even if the list of credentials stored on the device were to keep expanding.

3. Secure: The solution must be at least as secure as the passwords.

If the solution is to be token-based, loss and theft-resistant measures are to be also considered during the design stage. Stajano proposes Pico, which is a token-based password manager...
that has the three main aspects listed above built-in. Apart from that, there are also additional usability related benefits, security considerations and attributes with more details presented in Pico’s whitepaper.

The core design of Pico shown in Figure 6 is such that it has a camera that is capable of scanning 2D visual codes [10]. Communication between a computer and Pico happens over a radio. The thousands of memory slots within Pico are responsible for storing the credentials (private keys) of different apps, for instance, Gmail, Facebook and so on. The web page that the user wants to log on to presents a 2D visual code which is a self-signed certificate containing the public key of the app. A plugin is responsible for generating the 2D content and interacting with the web page, and it needs be installed on the browser. Pico then challenges the web app to prove its authenticity by signing a message with the correlated private key. If the web app successfully decrypts the message and responds back to Pico, a session gets established, and the user gets logged in.

Another feature of Pico is Picosiblings. These are devices that are always supposed to be in the vicinity of the Pico device, which implies the physical presence of the user. They could be everyday devices like key fobs, mobile phones, PDAs, etc. The concept is to automatically lock Pico in the absence of Picosiblings and unlock otherwise. This is to reduce the number of times a user would need to manually unlock the Pico. Stajano also proposes a docking station for the Pico, which would allow recharging and backing up of the device to the storage present within the docking station.
A state diagram of Pico is also presented in Figure 7. When a user buys Pico for the first time, the storage within it has no data. The user can either choose to restore from a backup or manually start creating the login credentials by following the guide. Pico is in an unlocked state when it is in the presence of defined Picosiblings. Otherwise, it is locked after a timeout. The main and pairing buttons are responsible for manually unlocking the Pico. Sepukku refers to the destruction of data and resetting of the device back to its factory state.

It is to be noted that Pico’s hardware and software blueprints alongside its supporting materials are open-source, allowing anyone to copy and modify for their own needs. The device is currently not available for purchase.
2.7 PUF BASED PASSWORD MANAGEMENT

PUF stands for Physically Unclonable Functions, and they represent the uncontrollable and unpredictable manufacturing variations during the fabrication stage of Integrated Chips (ICs) [5]. Due to such variations, two ICs of the same model from the same manufacturer would produce different output responses for the same input at a micro-level. The variations are also tough to clone and difficult to predict during the manufacturing stage. Two main properties of PUFs are uniformity, which is the capability of generating 0’s and 1’s proportionally, and uniqueness, which is the attribute of producing unique responses for different ICs under the same design and input constraints. These intrinsic variations of PUFs can be taken advantage of and be used as a tool to protect password hashes.

Figure 8: PUF layer challenge vs response. Diagram recreated based on [5]. A PUF Layer is responsible for taking in the challenge and responding with a property based on inherent PUF characteristics. This response is used to strengthen the security of passwords.
In Figure 8, a tuple of (label, user password, domain) is passed to the conventional functions that are responsible for hashing the tuple and comparing the hashed value against the hash of credentials stored in the password database. But with the PUF Hardware layer, the tuple’s hash is taken as a challenge input, and the output that comes out of the PUF is compared against the record stored in the password database. This means that no two hardware would generate the same output response, which means for someone to attack PUF Layer, he/she would need to have physical access to that device. It is also to be noted that the PUF layer requires extensive modification to the actual hardware by purchasing add on ICs such as Xilinx Zynq-7000 CPU. The system developer would also need to write drivers on the Windows/Linux/MacOS platforms to interact with the PUF layer [11].

For this to work, there also need to be significant alterations to the password database files both on local storage and on different servers. This is since, the conventional hash and salt pairs of user credentials are to be updated with the output responses of PUF. The advantage is that, even if an attacker somehow manages to obtain the user credentials through shoulder surfing, he/she would require access to the hardware.

One of the biggest questions arises when it comes to recovering passwords when the physical device gets lost. According to measurements done in the research paper, the cracking rate of PUF based outputs is 0 and the highest matching rate between the actual password and predicted password is only 23.3%. This no doubt brings a tremendous amount of security, but there is still work to be done in terms of loss-resistant measures.

In comparison to other password management mechanisms, PUF based technology brings more secure features but rates poorly in terms of usability and backwards compatibility (deployability).

Another research suggests the usage of SRAMs readily available in modern-day computers as PUFs [14]. This is to avoid the introduction of additional hardware for the PUF layer. The application layer for interacting with the PUF layer would have to rely on the APIs provided by the SRAM manufacturers in this case, whereas for a complete addition of a new layer, a system developer can define the API calls from the design stage.
This research argues that password managers usually synchronise the password vault (database) to remote servers, which allows for data breaching [11]. The extensions that come with modern-day web browsers to interact with the password vault also do not usually require the master password to be entered frequently. This can lead to unauthorised access or JavaScript-based hijacking attacks of the browser itself.

The high-level design of BluePass is presented as follows in Figure 9.

![High level protocol design for BluePass](image)

**Figure 9:** High level protocol design for BluePass. Diagram recreated based on [11]. Bluepass application on the Android device is responsible for sending the Site Password which is encrypted and can only be decrypted by the key retrieved from BluePass Server.

When a user wants to log in to a site, he/she would first need to authenticate to the BluePass server by supplying (1) Master Password. The BluePass servers then respond with (2) K2 if the Master Password is correct. The browser receives K2 and requests the mobile device by supplying the (3) Site Domain over Bluetooth connectivity. The Mobile Device responds with the (4) Encrypted Site Password. The browser makes use of K2 to (5) decrypt the Encrypted Site Password and access the site. Note that K1 and K2 are RSA key pairs which mean K2 can be used to decrypt the content that is encrypted using K1.

K2 is stored permanently on the computer browser if it is a trusted device. Otherwise, it is only stored temporarily and requires re-downloading from BluePass Server upon a timeout. This is possible because the device MAC address of the machine is also stored on BluePass Server. Since Bluetooth is used, the attacker needs to be in proximity to launch an attack. In
terms of recovery, BluePass does not store any password credentials on the server, but the password vault on the mobile device can be backed up to other mediums through the BluePass application. Since BluePass is reliant on Bluetooth connectivity, its absence on a computer would mean BluePass would not be usable.

BluePass rates high in terms of security but when it comes to deployability, it rates poorly because of the dependency on a running mobile device. In terms of usability, it rates fairly well.

2.9 SPHINX (RENAMED AS STOPs)

The idea of this research is to use a secondary device as a tool to compute a complex password from a simple password [19]. The intent is to allow a user to use a very simple password (pwd) such as “abc123”. Once this pwd is entered into the password field of the computer browser preceded by a special character such as @, the browser starts to send the pwd and domain of the website to the secondary device such as an Android based mobile. The SPHINX application on the mobile device is responsible for computing rwd, a pseudo randomised password with a high entropy value for the supplied domain. For each service, a unique key is stored on the secondary device to carry out the computation of rwd. This rwd replaces pwd in the password field and is sent to the server for authentication. To prevent Man in the Middle Attacks, SPHINX makes use of the FK-PTR protocol, which is an instantiation of OPRF (Oblivious Pseudo-random Function). This function generates rwd using key (k) such that the device D (Android app) learns nothing from the interaction, but C learns only the value of rwd, which is computed by the protocol. Hence, neither rwd nor pwd is stored on any devices. The diagram in the research paper provides a clear illustration of the entire process.

One drawback to this approach is that all existing passwords need to be updated with new rwds for different services. Since SPHINX relies on OPRF protocol, which can even be deployed on a Raspberry PI, the solution becomes rather flexible to implement. It needs not be a mobile device.

SPHINX ticks all the boxes in security, but when it comes to usability it rates as fair, and in terms of deploy-ability, it is very poor. Meaning, it is not backwards compatible at all.
Table 1: Summary of hardware and software-based password managers. Table displays the attributes and their strengths/weaknesses of each password management solution.

<table>
<thead>
<tr>
<th></th>
<th>Requires system level drivers+new hardware layer</th>
<th>Opensource</th>
<th>Requires application level software</th>
<th>Hardware based or Software based</th>
<th>Compatibility with previous configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PUF based</strong></td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
<td>Hardware</td>
<td>Not compatible</td>
</tr>
<tr>
<td><strong>SPHINX</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Hardware</td>
<td>Not compatible</td>
</tr>
<tr>
<td><strong>BluePass</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Hardware</td>
<td>Not compatible</td>
</tr>
<tr>
<td><strong>Pico</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Hardware</td>
<td>Compatible</td>
</tr>
<tr>
<td><strong>Steganographic</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Software</td>
<td>Compatible</td>
</tr>
<tr>
<td><strong>KeePass</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Software</td>
<td>Compatible</td>
</tr>
<tr>
<td><strong>Dashlane, 1Password, LastPass</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Software</td>
<td>Compatible</td>
</tr>
</tbody>
</table>

Table 1 presents a summarized version of all solutions presented in the literature review. Generally, only PUF based solutions would require new drivers because an extra chipset is added on top of existing machine configuration of the user. Solutions that make use of existing technologies such as wireless protocols do not require any additional drivers for existing operating systems. Software based applications require only the application level software. For those solutions that take on the task of generating long complex password for the user, existing database records will need to updated with new passwords, hence, it would affect backwards compatibility.
As presented in the introduction section, E2PM can be broken down into two drive partitions, the Core System and the Secured Data partition. The core system is a non-modifiable image during the run time, but the secured data partition allows for both reading and writing. The scientific method used for this research is experimentation which leads to creation of prototype software. For creating E2PM, the following specifications are used as shown in Figures 10, 11 and 12.

- **Host Machine**

  ![Host Machine Specifications](image)
  
  Figure 10: Host Machine Specifications used to create E2PM device.

- **Flash drive from Sony with 32 GB capacity.**

  ![32 GB Flash Drive](image)
  
  Figure 11: 32 GB flash drive from Sony easily bought online or in electronics store. E2PM works with any other similar flash drive.
• Ubuntu Mini Remix-16.04.2-amd64.iso ¹ (295 MB)

![Ubuntu Mini Remix 16.04 LTS 64bit download](http://www.pcds.fi/downloads/operatingsystem/ubuntubased/miniremix/latest/miniremix.latest.html)

Figure 12: Ubuntu Remix 16.04 LTS 64 bit

• VirtualBox Application v6.1.26_Ubuntu r145957 with VBox-Manage installed.

VirtualBox² is a powerful virtualization platform for x86 and AMD64/Intel64 processor-based machines available for both enterprise and personal use. VBoxManage tool is an additional product that needs to be installed for interacting with VirtualBox through the command-line interface. It provides advanced features such as creation of raw disks, which are not directly available through the Graphical User Interface.

It is to be noted that E2PM will work on any Intel64 and AMD64 processor based systems. For devices such as Chromebooks, E2PM will work through live boot only, since VirtualBox is not available. This is similar for other lite systems where VirtualBox is unavailable.

### 3.1 Core System

The Ubuntu Mini Remix iso file needs to be mounted first to build the core system. The file contents present in the iso are as follows.

```
aungnaingoo@N92:/media/aungnaingoo/Ubuntu-amd64-16.04$ tree -L 2 -lrtha .
```

---


² [https://www.virtualbox.org](https://www.virtualbox.org)
The file that needs to be customised for E2PM is filesystem.squashfs. This file is the largest in the iso package and has a size of around 260 Megabytes as shown in Figure 13. It contains all user space applications needed to run in a cli based environment.

To prepare the environment for customization, a working directory needs to be created. Within the working directory, all contents from the iso, including the hidden files and folders such as .disk, need to be copied over recursively. Hence it is essential to make the hidden files visible beforehand in the host system.

3.2 UNPACKING AND PREPARING FILE SYSTEM FROM ISO

The prerequisite before continuing is to install the required software by executing as follows.

```
$ sudo apt-get install squashfs-tools genisoimage syslinux xnest
```

This will install squashfs-tools for squashing and unsquashing filesystems. Genisoimage is used for creating ISO9660 files. SYSLINUX is required to install syslinux bootloader on a FAT
filesystem. Xnest (optional) is needed to load the applications from filesystem.squashfs on the host system in an X windows based client and server set up.

To unpack filesystem.squashfs, the following commands are used.

```bash
$ sudo unsquashfs filesystem.squashfs
```

This will unpack the file system from the iso into a folder called squashfs-root. This is where all the following work is done. Once that is executed successfully, the file system can be modified as one wishes and then repacked back into the iso.

Some of the files from the host system need to be temporarily mounted at this point. They are /proc, /sys and /dev. The following commands are executed.

```bash
$ sudo mount -t proc /proc proc/
$ sudo mount -t sysfs /sys sys/
$ sudo mount -o bind /dev/ dev/
```

This will allow the fake root file system to fully function on the host without any errors.

The networking configuration file from the host system also needs to be copied over to allow networking so that packages can be installed.

The following command is used.

```bash
$ sudo cp /etc/resolv.conf etc/
```

### 3.3 Modifying File System

To modify the file system, chroot command is used. It is a potent tool used to create fake root environments for testing, recovering system files and reinstalling bootloaders [1]. Once the fake root file system is chroot-ed into, the system cannot modify files outside of this new fake root file hierarchy. This is also known as “chroot jail”. The following command is executed to chroot into the Ubuntu remix’s squash-fs root directory.

```bash
$ sudo chroot . bash
```

This will explicitly run a bash shell after chroot-ing into the fake root environment, which will be addressed as E2PM environment from now on.
At this point, some valuable packages can be installed using the usual apt-get commands. The packages to be installed are python3, python3-pip, Midori web browser, lighdm and Vim. Once python3-pip is installed, pip package manager can be used to install Bottle \(^3\) and Pycryptodomex \(^4\).

Bottle is used for creating simple and lightweight micro-web server applications based on Python. Web routes can be easily defined and exposed on functions which can then be accessed through HTTP requests. Pycryptodomex is used for creating a custom program responsible for encryption and decryption of user credentials which will be put in a safe vault located in a secured data partition. This library is chosen because it does not depend on external dependencies like OpenSSL and is purely written in Python with only a few performances critical pieces of code written in C extensions.

At this point, it is important to highlight that the choice of Python, Bottle, Midori and all other open source applications mentioned is due to limited time frame, prototypical nature of the project, and for convenience of the developer. There are several other ways that the password manager can be developed to leave a smaller file size. But essentially, the steps are roughly the same for creating the filesystem.squashfs image.

\(^3\) https://bottlepy.org/docs/dev/
\(^4\) https://pypi.org/project/pycryptodomex/
3.4 CUSTOM PASSWORD MANAGER PROTOTYPE

```python
from Cryptodome.Cipher import AES
from base64 import b64encode, b64decode
import json

class Naing:
    def __init__(self, key):
        self.password_file_name = "e2pm/savedpasswords"
        self.key = str.encode(key)
        self.iv = ""
        self.encryption_obj = object()
        self.decryption_obj = object()
        self.retrieved_passwords = []

    def save(self, password, tag):
        self.encryption_obj = AES.new(self.key, AES.MODE_CFB)
        password_raw = str(password)
        encrypted_string_bytes = self.encryption_obj.encrypt(password_raw.encode('utf-8'))
        encrypted_string = b64encode(encrypted_string_bytes).decode('utf-8')
        iv = b64encode(self.encryption_obj.iv).decode('utf-8')
        self._writeToFile({"iv": iv, 'ciphertext': encrypted_string, 'tag': tag})
        del self.encryption_obj

    def readFromFile(self):
        tag_pass_list = []
        with open(self.password_file_name, "r", encoding="utf8", errors='ignore') as file:
            self.retrieved_passwords = file.read().splitlines()
        for entry in self.retrieved_passwords:
            b64 = json.loads(entry)
            iv = b64decode(b64['iv'])
            ct = b64decode(b64['ciphertext'])
            tag = b64['tag']
            self.decryption_obj = AES.new(self.key, AES.MODE_CFB, iv=iv) # CFB mode
            pt = self.decryption_obj.decrypt(ct)
            tag_pass_list.append(f"{tag} : {pt.decode('utf-8')}"
        del self.decryption_obj

        return tag_pass_list

    def _writeToFile(self, password_entry):
```

3.4 CUSTOM PASSWORD MANAGER PROTOTYPE
FILENAME: naiing.py

To summarize, this program uses pycryptodomex to encrypt or decrypt in AES-256 mode. It will take in a master password of 16 characters in length which must be remembered by the user. This key will be used by the AES algorithm to encrypt the password entries made by the user.

To interact with naiing.py, a Bottle based server prototype is created as follows.
@app.route('/e2pm/save', method='POST')
def save():
    password = request.forms.get('password')
    tag = request.forms.get('tag')
    app.pwm.save(password, tag)

@app.route('/e2pm/stop', method='GET')
def stop():
    return 'Application stopped'

if __name__ == '__main__':
    run(app=app, host='localhost', port=8080, debug=True)

• Filename: startup.py
• File attributes: -rwxr-xr-x

This application is responsible for creating an instance of the password manager and taking input from the user through HTTP, but only on localhost. This is obviously for security reasons to prevent other machines in the local area network from accessing this application on a defined port (8080 in this case).

To start off this application, it is required to run this application when the user space loads. This can be done by creating an entry in squashfs-root/etc/init.d directory. The content is as follows.
#!/bin/sh
mkdir /e
for i in $(seq 1 5); do
  if [ -e /dev/disk/by-label/E2PM ]; then
    echo "persistent storage found"
    sleep 1
    mount -t ext4 /dev/disk/by-label/E2PM /e
    break
  else
    echo "Waiting for persistent storage (try $i)"
    sleep 1
  fi
done
sudo python3 /startup.py &
sudo iptables -A OUTPUT -p all -m owner --gid-owner root -j DROP

• Filename: S50start
• File attributes: -rwxr-xr-x

The main idea is to create a directory called e2pm in root and then check for a disk partition with a label named E2PM. This is the secured data partition that is available for read/write operations. Once this partition is detected, the partition will be mounted under /e2pm. The bottle-based server script is started at the end of this file in the background. Internet is disabled by dropping all IP packets for the root group, which is the only group available.

Once done, the temporarily mounted host system files can be dismounted using the following commands.

$ sudo umount -l proc/
$ sudo umount -l dev
$ sudo umount -l sys/

To exit e2pm filesystem, it can be done simply by executing exit command. This will switch the system back to the default root file system on the host.
3.5 SQUASHING BACK THE FILESYSTEM

To squash back squashfs-root directory, the following command is used.

```sh
$ sudo mksquashfs squashfs-root/ filesystem.squashfs -comp xz
```

It takes a few minutes, depending on the number of tools and software installed in the e2pm file system. It is recommended to keep the size below 2 GB. In this paper, the resulting file size turned out to be 1011 MB which is well below 2 GB. The new filesystem.squashfs is to be overwritten to iso/casper folder in the working directory of e2pm system.

The filesystem.size file under iso/casper also needs to be updated and this can be done by executing the following command.

```sh
$ sudo printf $(sudo du -sx --block-size=1 squashfs-root | cut -f1 ) > iso/casper/filesystem.size
```

By this point, the e2pm system is ready to be repackaged back into a new iso file.

3.6 E2PM ISO AND FLASHING

To create the E2PM iso, the following is executed inside the <working directory>/iso folder is used.

```sh
$ sudo mkisofs -D -r -cache-inodes -J -l -b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -o ../E2PM.iso
```

This will create an iso file titled E2PM.iso in the root of the current working directory, indicated by a full stop at the end of the command. It takes a while to create the resulting file based on the host machine’s resources. The file size in this paper turned out to be 1.1 GB in size. A program such as Rufus shown in Figure 14 can be used to flash the iso on the drive. It will also install any missing system files required to boot directly from the drive. The only downside of using Rufus is that it runs only on a Windows-based machine.

---

5 https://rufus.ie/en/
3.7 Preparing the Sony 32GB Drive

The USB drive’s details can be found by executing `lsblk` program. In this paper, it is `/dev/sdb`. A program such as `cfdisk` can be used to resize the disk and create another partition with remaining size, given that at least 1.5 to 2 GB needs to be left untouched as the core system is already allocated. The secured data partition (`/dev/sdb2`) then needs to be labeled as E2PM since the $5@0 script and the password manager relies on this label. Its format needs to be `ext4`. This can be done by executing the following commands.

```
$ sudo mkfs.ext4 /dev/sdb2
$ sudo e2label /dev/sdb2 E2PM
```

At this point, the host system can be booted directly from the flash drive after setting the order of boot device in BIOS depending on the manufacturer’s instructions.
Figure 15: E2PM screens from live boot session displaying different interfaces such as during login, loading of password manager and accessing it as well as the encrypted passwords.

Figure 15 shows the different screens when E2PM is loaded. After bypassing the login screen using the default username “ubuntu” and password “” (blank), Midori web browser is loaded. The route defined by localhost:8080/e2pm asks the user to enter a new password or reuse a previous password. This is the 16 characters master key. If the correct key is supplied, E2PM manages to decrypt all the passwords and corresponding tags stored in the secured vault. New tags and corresponding passwords can be stored in the main URL once a master key is successfully created or reused. The system has no internet access, and hence, there is no way for an attacker to target E2PM over the network.
Once the usage is done, E2PM can be powered off and the USB drive can be ejected safely. For backing up the password vault, it can be easily done by accessing the E2PM secured vault partition and manually copying over the encrypted files and storing them in other secured mediums. The encrypted files are just normal text files with encrypted credentials in JSON format.

### 3.9 Configuration for Access Through Virtual Box

A generic 64-bit Linux based virtual machine can be created with 3GB memory and 1 core processor.

To allow for E2PM’s access directly from VirtualBox, the USB drive needs to be set as a raw disk so that VirtualBox directly interacts with the contents on the USB device instead of creating any intermediary image files [12]. VBoxManage is used at this point to execute and create the following files.

```bash
# Create the rawdisk for core system
$ sudo vboxmanage internalcommands createrawvmdk -filename ~/$cryptousb.vmdk -rawdisk /dev/sdb1

# Change the ownership to oneself
$ sudo chown aungnaingoo:aungnaingoo cryptousb.vmdk

# Change the file permission for read, write and execute
$ chmod u+rwx,g+rwx,o+r cryptousb.vmdk

# Create the rawdisk for secured data partition
$ sudo vboxmanage internalcommands createrawvmdk -filename ~/$cryptousbdata.vmdk -rawdisk /dev/sdb2

# Change the ownership to oneself
$ sudo chown aungnaingoo:aungnaingoo cryptousbdata.vmdk

# Change the file permission for read, write and execute
$ chmod u+rwx,g+rwx,o+r cryptousbdata.vmdk
```

The details of the resulting files are as follows.

```
-rwxrwxr-x 1 aungnaingoo aungnaingoo 629 Feb 24 17:15 cryptousb.vmdk
-rwxrwxr-x 1 aungnaingoo aungnaingoo 630 Mar 28 16:56 cryptousbdata.vmdk
```

Notice the tiny file sizes. These are interface files rather than image files. The two files can be loaded into VirtualBox manager with the USB drive connected.
Figure 16: E2PM screens from VirtualBox session. VirtualBox needs to be configured so that the raw USB can be accessed directly from the host machine.

Once the settings are completed, E2PM can be directly started from VirtualBox and interacted through the Midori browser in the guest OS as shown in Figure 16. New entries will be directly appended to the secured file through the password manager program. The read and write status can be intuitively observed by monitoring the LED status on the USB drive. Usually, it takes a few seconds before VirtualBox writes the new entries to the drive. E2PM is only accessible through VirtualBox if the drive containing the E2PM system is plugged in.
4 RESULTS

4.1 ASSESSMENT OF E2PM BASED ON USABILITY, DEPLOYABILITY, AND SECURITY

Figure 17: Rough assessment of E2PM in terms of Usability, Security and Deploy-ability. High scores for Security and Deploy-ability, but a weak score for usability as it requires some technical knowledge.

As shown in Figure 17, E2PM, from a usability perspective, is not very easy to use and requires a little technological knowledge. Also, there are no copy-paste options for the user, so they will have to manually type the revealed passwords on the host system. Hence, this is an area that can be improved.

From a deploy-ability perspective, unlike other hardware-based password managers, E2PM is straightforward to deploy. The only requirement is to have a USB port and VirtualBox installed and configured if a user wishes to access E2PM on the host machine. Other strong password managers such as KeePass as shown in Figure 18 can also be installed and configured on the Core System, and the secured credential file can be placed in the persistent partition of E2PM.

From a security perspective, E2PM uses a master key of 16 characters in length, hence resulting in a very high entropy.
The password credentials are encrypted using the AES-256 algorithm through the pycryptodomex library. This is the state-of-the-art security standard and is impossible to break as of the present.

Figure 18: KeePass running on E2PM connected through Virtual Box. This is one way of accessing E2PM without requiring a user to reboot the device.
Table 2: E2PM’s entry among other solutions. Displays how E2PM rates as compared to other existing solutions on the market and from the Literature Review.

As shown above, in Table 2, E2PM does not require any special drivers or hardware layers to be installed on the host system. It is fully open-source since it makes use of open-source software. The password manager is already built into it. Hence, no other tool must be installed on the host system except for VirtualBox and its related configurations if a user wants to use E2PM through it. It is fully backwards compatible since a user would not need to change their account credentials anywhere else to fit the needs of E2PM.
This research has presented a highly secure way of storing password credentials on an external device in a completely enclosed manner. The core system on E2PM does not have networking enabled; hence there is no way for an attacker to launch an attack on it over the web or through other protocols such as Bluetooth. E2PM can be booted directly from the USB or through the VirtualBox application. The attack vectors when directly booting from the USB are very much narrowed down, with the only downside being that the user will have to temporarily remember the displayed passwords on the Midori browser. Key loggers are ineffective in this case because firstly, an attacker would need physical access to the machine’s USB ports, and secondly, the kernel is created to support only a limited number of devices and may not be able to detect a key logger. The kernel can be customized or swapped to completely drop all unnecessary modules (drivers) depending on the required constraints.

However, when running E2PM through VirtualBox, the security measures on the host machine are more critical. But still, it will be a challenging task for an attacker to access the USB device. Even if they successfully obtain access to the credential files through physical theft, the 16 character master key length would not be breakable with current limitations in computing resources. The host machine could be connected to the internet, and hence, it is the main reason why E2PM is configured such that the E2PM’s URL routes are not accessible through the host machine. This is done when defining the parameter as ‘localhost’ during the start of the Bottle-based server script. The user will have to refer to the VirtualBox guest’s screen to see the passwords and type on the host. Essentially, it is a trade off between usability and security.

Another form of attack that every hardware can be prone to is Cold Boot attacks. It is a technique where the physical RAM (Random Access Memory) is plugged out, and its contents are freshly dumped on the attacker’s machine for further analysis using computer forensics tools [6]. Although it requires a large
amount of time and effort, depending on the size of physical RAM, this attack is not so common because of the physical nature of RAM. Hence, to find the master key in the RAM, it is akin to looking for a needle in the haystack. Users are strongly encouraged to secure their devices and turn off the computer and unplug E2PM from USB slot when not in use. This will decrease the likelihood of Cold Boot attacks. E2PM, like any other technology that displays on a screen, can still be exposed to other forms of non-technical attacks such as shoulder surfing, screen capture, taking photos through a phone and so on. However, there are already protection mechanisms such as privacy shields for the computer screen that counter such attacks.

As for storage, at least 2 GB needs to be allocated for the core system, and the remaining can be utilized for the persistent secure partition. In this paper, out of 32 GB, at least 26 GB was available for storage, which is plentiful even for enterprise usage.

E2PM does not provide any backup features yet. It is for the user to manually back up the credential file from the secured data partition on the drive. From a usability perspective, this is indeed a drawback which needs to be addressed in future work. In terms of security of the master key, it is never stored anywhere on E2PM. So if a user loses it, there is no way for them to recover their credentials. The security of the credential database is protected by AES-256 encryption which is practically unbreakable. In terms of security of communication to and from E2PM, the live boot provides better security due to the limited number of communication features enabled. Through VirtualBox, the security of the host machine is critical. If the host machine is compromised through some form of malware (such as malware that takes screenshots and sends over to third parties), it can compromise the output screen of E2PM. Hence, it is important that a user takes responsibility of installing anti-virus software and keeps the virus definitions up to date.

E2PM can still be prone to offline attacks where a malicious user manages to physically obtain the device. For instance, a malicious user can steal a device, unsquash the filesystem, install a keylogger, resquash and then place it back at the same place. The owner of the device would use it as per normal while
their keystrokes get recorded and placed in the secured partition on E2PM. The malicious user can then steal the device again and find the file containing the keystrokes. This can lead to the compromise of the master key. Therefore, it is important that a user never leaves their E2PM alone when outside of their trusted environment. In future work, a checksum calculator can be developed to calculate the integrity of files on E2PM’s read only drive. Any deviations would then be flagged out as anomalies, and the user would then be able to safeguard against attacks on their master keys.

Due to limited timing, it has not been possible to practically analyze all vectors of attacks extensively. Also, a usability study needs to be done to understand the aspects that need to be improved so that an everyday user can use E2PM comfortably. As such, these are left for more comprehensive work in the future. The essence of E2PM has been presented in detail alongside step by step instructions so that any (reasonably technical) reader can follow and create their own version of E2PM.
CONCLUSION

This research started to bridge the gap between hardware and software-based password managers. The challenge with lots of hardware-based password managers is that they are not backwards compatible, which means, almost always, a new hardware layer is introduced, and additional drivers have to be written for supporting a user’s Operating System and device. If no new hardware layer is introduced, and instead, existing breakout devices such as mobile phones are used to store or compute new passwords, it relies on the availability of that breakout device and the wired or wireless interaction between the device and the computer. If such interaction media is not available, the user would not be able to use the password manager. If new computations of passwords are generated, the accounts that the user already has needs to be updated everywhere. There is also much work that needs to be done to improve the backing up of passwords in case of loss of the hardware-based password manager.

The challenge with software-based password managers is that they store the password files on the user’s machine or on a cloud of a third party server. This increases the likelihood of data leakage when a virus infects the user’s computer or when a hacker breaches into the third party servers. Moreover, most third-party service providers do not disclose how they store the passwords on the server and what kind of processing happens on top of those files.

To answer the three main research questions from Chapter 2, E2PM comes in to bridge the gap by providing a way to store passwords safely on a separate device in a secured manner and minimize its exposure to the web and host machine’s potential anomalies. There is also no loss of backwards compatibility as compared to existing hardware based solutions because E2PM allows the user to save and recall existing passwords into E2PM without resetting their passwords to fit the needs of other solutions. In terms of backing up, the user can always choose to temporarily copy and store the password file in another USB drive. E2PM can either be booted directly or used through VirtualBox.
application. Hence, it eliminates any need for additional drivers and changes to the hardware layer of the existing machine. The idea of E2PM is to make use of existing open technologies and integrate them to act as a highly compatible hardware-based password manager, which is at the exact time cost-effective.

E2PM, by no means, is a complete solution, yet, as more practical analysis needs to be conducted alongside usability studies. The usage of technologies can also be revamped by making use of another technology stack to build the password management layer. However, the essence of E2PM would remain the same, which is to provide a self-enclosed portable hardware-based password manager that is low cost and backwards compatible, where the user is in full control of their credentials without the passwords ever touching the host system’s hard disk.


