

**Universidade do Minho**

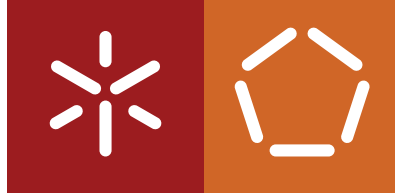
Escola de Engenharia

Departamento de Informática

João Nuno Alves Lopes

**Implementation of practical  
and secure methods for storage of  
cryptographic keys in applications**

February 2022



**Universidade do Minho**  
Escola de Engenharia  
Departamento de Informática

João Nuno Alves Lopes

**Implementation of practical  
and secure methods for storage of  
cryptographic keys in applications**

Integrated Master's in Informatics Engineering

Dissertation supervised by  
**Rui Oliveira**  
**Ana Nunes Alonso**

February 2022

---

## COPYRIGHT AND TERMS OF USE FOR THIRD PARTY WORK

---

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

LICENSE GRANTED TO USERS OF THIS WORK:



**CC BY**

<https://creativecommons.org/licenses/by/4.0/>

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos. Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada. Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

---

## STATEMENT OF INTEGRITY

---

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

---

## ABSTRACT

---

Encryption has been essential to protect modern systems and services. It became the security foundation of databases, payment systems, cloud services, and others. Cryptography enabled the creation and validation of digital signatures, where the protection of the private key is very important to prevent false signatures. Cryptocurrencies rely on this mechanism.

Crypto wallets hold private keys used to sign transactions and prove ownership of a digital asset. These have to keep the private key secure, but accessible to its owner, as it may be needed frequently. With the increasing number of decentralized web applications that interact with a blockchain, this subject has become more prevalent, as they usually require frequent signatures from the user.

The mass adoption of cryptocurrencies by non-technical users urged the creation of crypto wallets that are secure but prioritize usability. Some of these are hosted services that store the private keys in their servers and others are non-hosted, where the user is responsible for storing it. When implemented as a browser plugin, these wallets allow the user to seamlessly interact with a web application. The rise of cloud technology brought forth multi-signature on the cloud, by combining different cloud services owned by the user. These give the user control of his private key and are less vulnerable to cyber attacks.

In this work, it is presented a comprehensive analysis of existing crypto wallet approaches in usability and security to understand the existing problems. The next step was to propose multiple possible solutions to those problems and produce their implementations. These take advantage of previously studied multi-cloud technology and are used to attempt to improve usability and security. To evaluate the proposed solutions and to compare them to the existing ones, we have developed a framework that consisted of various objective tests based on previous work, which have the goal of evaluating security and usability.

Finally, the proposed and existing solutions were compared using the proposed framework.

**KEYWORDS** data encryption, multi-cloud, crypto wallet.

---

## RESUMO

---

A encriptação tem sido fundamental para proteger serviços e sistemas modernos, tornando-se essencial para proteger bases de dados, sistemas de pagamento, serviços de nuvem, entre outros. A criptografia permitiu a criação e validação de assinaturas digitais, onde a proteção da chave privada é bastante importante para impedir assinaturas falsas. As criptomoedas dependem deste mecanismo.

As carteiras digitais contém chaves privadas que são usadas para assinar transações e provar a posse de um artefacto digital. Estas guardam a chave privada de forma segura e acessível ao seu proprietário, pois pode ser necessária frequentemente. Com o aumento do número de aplicações *web* descentralizadas que interagem com *blockchains*, este assunto tem ganho relevância, pois estas podem necessitar assinaturas frequentes por parte do utilizador.

A adoção em massa de criptomoedas por utilizadores não técnicos levou à necessidade de criar carteiras digitais seguras, mas que priorizam a usabilidade. Algumas destas carteiras classificam-se como serviços *hosted*, que guardam as chaves privadas nos seus servidores, e outras como *non-hosted*, onde o utilizador é responsável por as guardar. Quando implementadas como um *browser plugin*, estas carteiras permitem o utilizador interagir fluidamente com uma aplicação *web*. A ascensão da tecnologia nuvem permitiu o aparecimento da múltipla assinatura na nuvem, através da combinação de diferentes serviços de nuvem já possuídos pelo utilizador. Estes dão ao utilizador controlo total da sua chave privada e são menos vulneráveis a ciberataques.

Neste trabalho, é feita uma análise compreensiva à usabilidade e segurança dos vários tipos de carteiras digitais, para perceber os seus problemas existentes. O próximo passo foi propor várias possíveis soluções aos problemas encontrados e também fazer a sua implementação. Estas utilizam trabalho previamente estudado sobre tecnologia multi-nuvem e são usadas para com intuito de melhorar a usabilidade e segurança. Para fazer uma avaliação das soluções propostas e compará-las a produtos existentes, desenvolvemos uma *framework* para testar segurança e usabilidade com vários testes objetivos, baseados em trabalho previamente estudado.

No final, as soluções propostas e as já existentes foram comparadas utilizando o *framework* proposto.

**PALAVRAS-CHAVE**    encriptação de dados, multi-nuvem, carteira digital

---

# CONTENTS

---

## Contents [d](#)

### I INTRODUCTORY MATERIAL

#### 1 INTRODUCTION [4](#)

#### 2 BACKGROUND [6](#)

- 2.1 Symmetric algorithms [6](#)
  - 2.1.1 DES and TripleDES [7](#)
  - 2.1.2 AES [7](#)
  - 2.1.3 TwoFish [8](#)
  - 2.1.4 One Time Pad and Stream Ciphers [8](#)
- 2.2 Asymmetric algorithms [8](#)
  - 2.2.1 RSA [9](#)
  - 2.2.2 ECC [10](#)
  - 2.2.3 Overview [10](#)
- 2.3 Cryptographic key management [11](#)
  - 2.3.1 Hardware Security Module [11](#)
  - 2.3.2 Key Management Service [11](#)
  - 2.3.3 Cloud HSM [12](#)
  - 2.3.4 Other management strategies [12](#)
  - 2.3.5 Shamir's Secret Sharing with multi-signature [13](#)
  - 2.3.6 Summary [13](#)

#### 3 STATE OF THE ART [14](#)

- 3.1 Web Wallets [14](#)
  - 3.1.1 Hosted [14](#)
  - 3.1.2 Non-Hosted [15](#)
  - 3.1.3 Software wallets [16](#)
- 3.2 Hardware wallets [17](#)
- 3.3 Multisig with multiple devices [18](#)
- 3.4 Multi-Cloud wallets [22](#)
- 3.5 Discussion [23](#)

## II CORE OF THE DISSERTATION

<b>4</b>	<b>APPROACH</b>	<b>25</b>
4.1	Multi-cloud approaches	25
4.1.1	Server authentication	26
4.1.2	Secure enclave	26
4.1.3	Password manager	27
4.2	Other approaches	28
4.2.1	Centralized Server	28
4.2.2	Web local storage	29
4.2.3	Hardware wallet	29
4.3	Proposed architecture	30
<b>5</b>	<b>IMPLEMENTATION</b>	<b>32</b>
5.1	Implementation strategy	32
5.2	Wallet code architecture	33
5.3	Multi-cloud connect platform	33
5.3.1	Multi-cloud with server approach	34
5.3.2	Multi-cloud with password manager approach	34
5.3.3	Multi-cloud with secure enclave	36
5.3.4	Conclusion	38
<b>6</b>	<b>EVALUATION</b>	<b>39</b>
6.1	Usability	39
6.1.1	Number of clicks	40
6.1.2	Credential management effort	40
6.1.3	Discussion	42
6.2	Security	44
6.2.1	Memory Exfiltration	46
6.2.2	Stealing credentials	46
6.2.3	Overall conclusions on wallet security	48
6.3	Combined usability and security assessment	48
<b>7</b>	<b>CONCLUSIONS AND FUTURE WORK</b>	<b>50</b>



---

## LIST OF FIGURES

---

Figure 1	Symmetric encryption	7
Figure 2	Confidential communication	9
Figure 3	Asymmetric encryption with authenticated communication	10
Figure 4	Hosted web wallets	15
Figure 5	Non-hosted web wallets	16
Figure 6	Plugin non-hosted web wallets	17
Figure 7	Software wallets	18
Figure 8	Hardware wallets	19
Figure 9	Paper wallets	19
Figure 10	Multi signature wallets representation	20
Figure 11	Multi signature wallet with multi devices	21
Figure 12	Multi signature wallet with multi devices unlocking	21
Figure 13	Multisig wallet with multiple cloud storage providers	22
Figure 14	Server authentication architecture	27
Figure 15	Secure enclave architecture	28
Figure 16	Password manager architecture	29
Figure 17	Architecture	31
Figure 18	Code architecture	34
Figure 19	Import account from multi-cloud platform	35
Figure 20	Sign transaction on multi-cloud platform	35
Figure 21	Sequence diagram for the multi-cloud with server approach	36
Figure 22	Sequence diagram for the multi-cloud with password manager approach	37
Figure 23	Sequence diagram for the multi-cloud with secure enclave approach	37
Figure 24	Credentials, device compatibility and private key ownership for different approaches	40
Figure 25	Number of clicks and keyboard input	41
Figure 26	Combined effect of password length and pronounceability on the credential management effort (lower is better in usability)	42
Figure 27	Credential management effort for each approach (lower is better)	43
Figure 28	Combined user effort when signing a transaction for each approach (lower is better)	44
Figure 29	Possible attack vectors and approach vulnerability, along with potential damage and whether mitigation is possible	45
Figure 30	Approach vulnerability to different techniques	47

Figure 31	Required credential compromise and access for successfully exploiting each approach	48
-----------	---	----

Part I

INTRODUCTORY MATERIAL

---

## INTRODUCTION

---

Encryption has become one of the most fundamental tools of privacy and security of many essential services that are used in the world. It has evolved from being just a way to protect communications to be the cornerstone of protecting databases, payment systems, cloud services, and others. Managing the cryptographic keys used in this process is a challenge at a personal or enterprise level. If exposed, they may compromise entire systems, so it is necessary to have them well secured as well as access to its owners. For this, multiple techniques try to balance security and usability (e.g. key management services, multi-signature, hardware modules, etc.).

A prevalent use case for these techniques is in crypto wallet management. It has been gaining increasing relevance with the appearance of cryptocurrencies and other crypto assets platforms. Crypto wallets are secured by a private key used to sign transactions and to prove crypto assets ownership. The key must remain as secure as possible, as anyone with access, has full control of the funds or assets held in the wallet. The accessibility of the private key is as important as its security because it may be needed to sign transactions regularly. This problem is valid in multiple contexts, like private information, passwords, or crypto wallets, that has been gaining increased relevance due to the increase of decentralized web applications and games that rely on integration with a blockchain (like Uniswap, OpenSea, etc.) . There are approaches that store the private keys in their servers, which results in more user-friendly products like web wallets. Other types of web wallets are non-hosted, where the user has access to the private key and is responsible for storing it safely. The enormous advantage of this approach is the great integration with blockchain web applications when implemented as a browser plugin, that allows a user to seamlessly interact with an application using the crypto wallet. On the other side of the spectrum, some wallets use completely different approaches and value security the most, targeted at more technical users. These are usually implemented as physical hardware devices that offer great security but at the risk of loss of the device or a hardware failure. These approaches are not very user-friendly and are mostly used to protect large amounts of funds, as these have complex security systems in place that may not be intuitive for a non-technical user. Also, to tackle single-device vulnerabilities, multi-signature (with multi-device) approaches separate the key into multiple devices, that alone cannot compromise the wallet.

The problem tackled in this work is that the crypto wallets with the most users (web-based) do not provide great security, and the most secure are not easy to use. This problem has become more prevalent with the mass adoption by non-technical users that is accelerating the market to find new approaches and alternatives. New concepts of crypto wallets try to make the management of funds as easy as possible because complicated systems do not appeal to the large majority of users. With the rise of cloud technology, an alternative is multi-signature on the cloud, by combining different cloud services owned by the user, as in (Pontes et al., 2017). As for

usability, it gives the user control of his private keys. As for security, this wallet is not vulnerable to common attack vectors and loss or theft of a device, as the different parts of the key are protected by the user's password of each cloud service and it is assumed that cloud services are unlikely to collide. However, to our knowledge, there is no implementation of a multi-cloud wallet that offers seamless interaction with blockchain web applications.

The goal of this work is to create a crypto wallet implementation that combines different positive aspects of different wallet implementations. The basis for our implementation is the multi-cloud technology, that could offer good security, usability and web app interaction. We will search for the best compromise between security and usability.

Finally, the contributions made in this work are:

- The implementation of various multi-cloud approaches as web non-hosted crypto wallets, that allows us to have a prototype of a multi-cloud wallet with web app interaction;
- The creation of a framework for the usability and security evaluation of a crypto wallet;
- The final evaluation of the proposed multi-cloud approaches and the comparison between them and the existent approaches, using the framework that was designed.

---

## BACKGROUND

---

Key-based encryption transforms plain text into cyphertext. Successful decryption reverses the process and requires knowing the key (Bhanot and Hans, 2015). A key is a string of bytes with a specific size, depending on the chosen algorithm (Raigoza and Jituri, 2016) and keeping it secure, i.e. private and persistent, is a fundamental concern.

As for types of input data, there are stream ciphers and block ciphers. The first one encrypts blocks of data of fixed size and the latter encrypts continuous streams of data one byte at a time. These have different levels of complexity and efficiency and serve different purposes. Block ciphers are used more often on databases, cloud services, digital signatures, hard disks, etc. (Singh and Kaur, 2015), while stream ciphers are used in multiple web security protocols.

Regarding keys, there are two types of algorithms: symmetric and asymmetric. Generally, these can provide similar levels of security but are very different in terms of computational power needed and inefficiency (Chandra et al., 2014b).

### 2.1 SYMMETRIC ALGORITHMS

Symmetric encryption uses the same key to encrypt and decrypt information, as seen in Figure 1. Only with the key can the transformation be reversed to read the data. This means the sender needs to share the key with the receiver, for the latter to read it. There are multiple algorithms to perform it, with different levels of complexity and security. Complexity can be evaluated by analyzing how long encryption and decryption take. As for security, there are known attacks that can determine the vulnerabilities of the algorithm. Brute force finds the key by trying every possible combination. So, longer keys can increase the time necessary to perform the attack and make it unfeasible.

The main use cases for these algorithms include storing passwords, sensitive information in databases, disk drive encryption, and others, where encryption and decryption take place in the same platform and in the same context, where there is no need for the key to be shared (Salama et al., 2009). As symmetric encryption/decryption is more efficient than asymmetric encryption/decryption, this can also be used after a shared key has already been established.

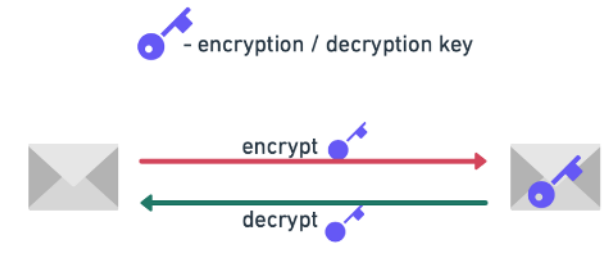


Figure 1: Symmetric encryption

### 2.1.1 DES and TripleDES

DES (Data Encryption Standard) is one of the oldest encryption algorithms. It was designed by IBM and published in 1974 (Coppersmith, 1994). It uses an encryption key with 56 bits and separates the input plain text in 64-bit blocks, returning also as output 64-bit blocks. This algorithm is not considered safe anymore due to the size of the encryption/decryption key, considered vulnerable to brute force attacks (Prasad et al., 2019). Also, the algorithm is very slow when implemented in software and more efficient with hardware implementations (Antonios et al., 2006), because it is mainly based on shuffling and substitution and has little computation involved (Srinivasa, 2015) (Thakur and Kumar, 2011).

TripleDES was designed to improve some of the biggest faults of DES by tripling the key size to 192 bits. This improved resistance to brute force attacks, which are still not possible with current machines. TripleDES wanted to replace its antecessor while preserving the existing software and equipment, so, it just divides the cryptographic key and applies the DES algorithm three times (Pich et al., 2018). This approach made the implementation more secure but tripled the encryption and decryption time which is a big disadvantage and makes it unusable in many use cases, where access to data requires very low latency (Kansal and Mittal, 2014).

### 2.1.2 AES

The Advanced Encryption Standard algorithm is widely used and came to replace Triple Data Encryption Standard. The algorithm can be used in three different variants that use 128-bit, 192-bit, or 256-bit key sizes (Cho et al., 2013). Security is improved with longer keys, but the computational cost of executing the encryption and decryption also increases. The block size that is used to divide the input is 128-bit and has 10, 12, or 14 rounds of permutations and substitutions, depending on the key size. It is very efficient both in hardware and software. AES outperformed DES and Triple DES completely both in security and in performance, that is decryption and encryption speed and the computational resources needed (Bhat et al., 2015) (Singh and Kinger, 2013).

### 2.1.3 TwoFish

TwoFish is an improvement over BlowFish, previous algorithm and was one of the candidates to become the international encryption standard, which was awarded to AES. It has also a 128-bit key starting size that is considered safe against brute force attacks (Nie and Zhang, 2009). The reason why AES was chosen as the standard is its better efficiency, especially in low power hardware, as seen in multiple tests while encrypting different types of data (Gaj and Chodowicz, 2000). TwoFish continues to be used by the cryptographic community, but it is far less generalized when compared with AES dominance.

### 2.1.4 One Time Pad and Stream Ciphers

One Time Pad (OTP) uses each bit of the cryptographic key to encrypt the corresponding bit of the plain text using the XOR operation. This algorithm is considered to be unbreakable if every condition mentioned next is met. First, the used key needs to be the same size or bigger than the plain text. Secondly, this cryptographic key needs to be completely random and never be repeated or reused. Similarly to other algorithms, the key cannot be exposed or revealed (Bellare, 2011).

Its biggest disadvantages are that communications of encrypted messages are impractical, due to the fact that the key is required to be different in every encryption process, so it is needed to pre-share the key in order for the recipient to decrypt the cyphertext (Matt and Maurer, 2013). Also, generating a random key the same size as the plain text may be very hard to achieve, depending on the environment where it is being used (Dodis and Spencer, 2002). One Time Pad was used mostly in the pre-computer era to encrypt information. It is not considered very practical to be used in modern systems due to the disadvantages pointed out, although it has a big advantage of being unbreakable (Rijmenants, 2009).

To tackle some of the mentioned problems of OTP, some stream cipher algorithms have been presented to improve usability. In this type of algorithms, a single key is used to produce a pseudo-random sequence of bits (Zeng et al., 1991). One of the most well-known stream ciphers is RC4, which has been broken in the past (Mantin and Shamir, 2002), but continues being used in the SSL, WEP, and WPA protocols, due to its simplicity (Gupta et al., 2014) (Kitsos et al., 2003).

## 2.2 ASYMMETRIC ALGORITHMS

Asymmetric encryption uses public and private keys. As the name suggests, the public key can be shared with anyone, unlike the private key that should not be exposed (Chandra et al., 2014a). Data is encrypted using the public key and can only be decrypted using the private key (Chandra et al., 2014b), as seen in Figure 2. The same can be done in reverse. Also, the private key is enough to generate the correspondent public key. This enables confidential communication without sharing secrets.

Imagine two people, Bob and Alice, that do not trust each other but want to exchange a message that needs to be authenticated by each one for it to be valid. Alice could use Bob's public key to encrypt the message and



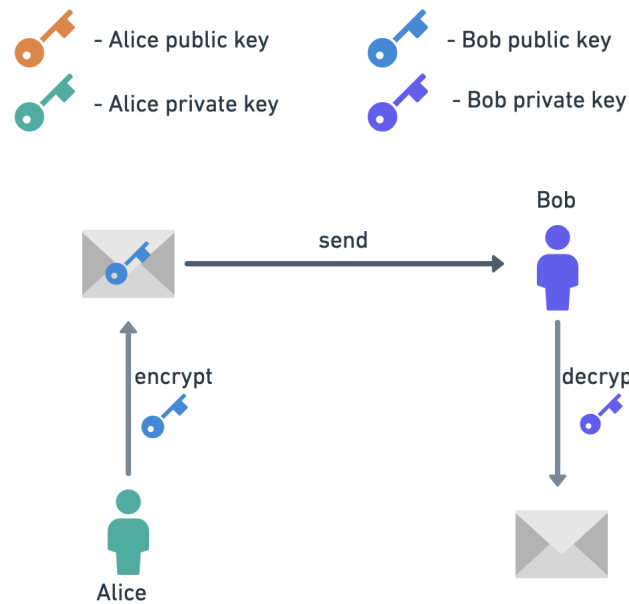


Figure 2: Confidential communication

send it. Then, Bob would use his private key to decrypt it. In this scenario, Alice knows for sure that the message sent can only be read by Bob, but he does not have proof that the message came from Alice.

To solve the authentication problem, Alice can sign the message with her own private key, and then encrypting it again with Bob's public key. When Bob receives the message, he can decrypt it with his private key and then verify it with Alice's public key. This way, he knows for sure who sent the message, and the sender cannot repudiate sending the message (Simmons, 1979). This example is portrayed in Figure 3.

The main use cases for asymmetric encryption include digital signatures (as seen in the example). It is also often combined with symmetric encryption in the SSL protocol (Hickman, 1994) that is used to encrypt communications over a network (Potlapally et al., 2002), and also to encrypt keys from symmetric encryption (Singh et al., 2016). Although this approach does not require sharing a secret key, secure storage of the private key is still an issue.

### 2.2.1 RSA

This algorithm generates the public and private keys based on two prime numbers (Milanov, 2009). It is based on the prime number factorization problem and is solved by finding two prime numbers knowing the result of their multiplication (Aufa et al., 2018). This becomes harder with larger numbers, which makes the algorithm more secure (Innokentievich and Vasilevich, 2017). Key sizes usually chosen are 2048-bit and 4096-bit, which are considered secure against brute force attacks.

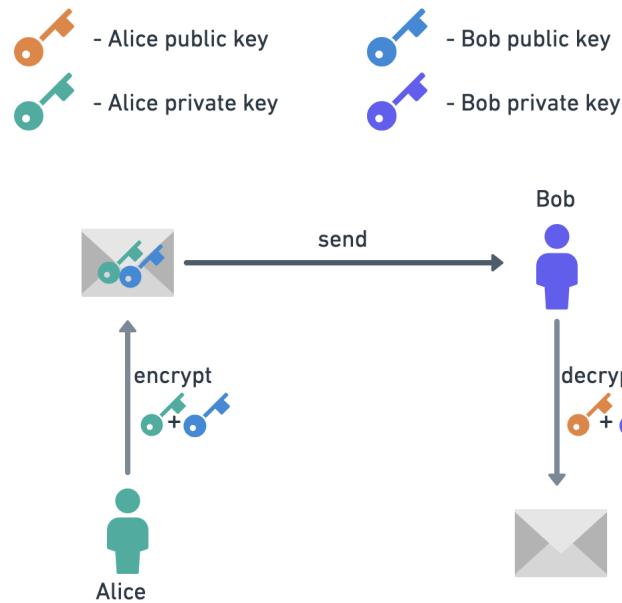


Figure 3: Asymmetric encryption with authenticated communication

In terms of performance, RSA decryption is slower than encryption, due to private keys often being larger than the public ones. It is usually slower than symmetric algorithms and requires more computational power (Hasib and Haque, 2008).

### 2.2.2 ECC

Elliptic Curve Cryptography algorithm is based on the elliptic curve discrete logarithm problem. ECC uses smaller keys than RSA, as it is usually implemented with 224-bit keys. It has been studied (Mahto and YADAV, 2017) that ECC outperforms RSA in decryption time in levels of security above, and including, RSA 2048-bit key (Gupta and Silakari, 2010). Although in encryption ECC is far slower, the overall time of encryption and decryption is much faster with larger inputs of data, as (Mahto and YADAV, 2017).

This may raise a question on why is ECC not the standard of asymmetric cryptography. This is mainly due to the fact that RSA was the first algorithm to be adopted (Milanov, 2009). Also, if the use case only requires a small input of data and the decryption time is not crucial, then RSA may be a better choice.

### 2.2.3 Overview

In symmetric encryption, AES variations of 128 and 256 bit are commonly used to encrypt passwords or bigger inputs of data.

In asymmetric encryption RSA and ECC are widely used. As it was mentioned above, RSA can be faster with small inputs of data, but it uses much larger keys. These algorithms also take a penalty in performance when compared with AES and TwoFish (Yassein et al., 2017) (Panda, 2016). Cryptocurrencies like Bitcoin (Nakamoto, 2009) and Ethereum, use the ECC algorithm to validate and generate digital signatures (Kikwai, 2017).

## 2.3 CRYPTOGRAPHIC KEY MANAGEMENT

As mentioned, the key and the private key used in symmetric and asymmetric encryption, respectively, must remain private in order for the encryption algorithms to serve their purpose, to protect data. To achieve this, these keys need to be stored securely but also need to be accessible, because the manager of the key might need it to encrypt or decrypt data at any time (Fumy and Landrock, 1993). So, there is the need to balance security and usability. In a scenario where the manager of the private keys needs to perform a decryption to read from a database, the key used to decrypt the information needs to be available at any time. But, if the manager of the key only needs it to sign transactions on a non-regular basis, the key may be stored more securely but also less available. Another important factor is whether keys need to always be accessible with low latency.

In all of these use cases, a single manager holds the ownership of the cryptographic keys. But what if these keys cannot be trusted by a single individual? This requires cryptographic strategies to manage the keys without having an individual with full ownership.

### 2.3.1 *Hardware Security Module*

Hardware Security Modules (HSM) are physical devices, that when combined with a server perform the storage of cryptographic keys, as well as key creation, encryption, and decryption. These devices are a very secure option to store private data, as they are internationally certified in different levels (Han et al., 2019).

The user can create new keys directly through the HSM or can store existing keys. Data can be encrypted without having to take the keys out of the module and its access is restricted and controlled by the user.

When using multiple HSM's, some problems like key deletions (Köppel and Neuhaus, 2013) and critical data loss can be solved by performing backups using mirroring techniques (de Souza et al., 2008). This helps make the data always accessible and consistent throughout the modules. Their main disadvantage is their hard maintenance (especially to update software) (Attridge, 2019). It is a very used approach in modern enterprises, but usually, as part of a major cloud service like it will be mentioned next.

### 2.3.2 *Key Management Service*

These services offer a different way of interacting with an HSM. Instead of a user or an enterprise having to buy and set up an HSM, these just subscribe to a KMS that stores the keys in a certified HSM owned by the service (Fumy and Landrock, 1993). A KMS offers a way of managing the keys of a client, as well as the same

properties that an HSM natively supports, like key creation and encryption and decryption of data (Beer and Holland, 2014).

The keys are protected by a master key that is used by the client or enterprise to access them. In some cases, this master key is used to encrypt other keys, creating a key hierarchy that minimizes the exposure of plain text (Huang and Chen, 2018). This means that the keys cannot be accessed by anyone except the master, as everything inside the service is encrypted and unrecoverable without the master key. The biggest disadvantage of this approach is requiring key sharing in a multiple ownership scenario.

High availability of the service and loss protection is usually assured, depending on the reliability of the provider. Although the stored keys cannot be exposed, their deletion is possible as a result of a cyber attack (AlZain et al., 2012).

As most of these services are backed up by cloud providers, they offer integration with applications and databases already being used on that cloud service. They also have advantages in terms of scalability and the low price paid on demand, when compared to the investment of multiple HSM's (Dowsley et al., 2016). This comes with reduced privacy on where the keys are stored, because a single HSM may be shared by multiple users. Cloud providers that offer these services include AWS (Amazon Web Services) (AWS, 2020b) and Google Cloud (Google, 2020b).

### 2.3.3 *Cloud HSM*

A Cloud HSM is another service very similar to the above-mentioned KMS. It is also offered by cloud services and is also used to store and manage keys in an HSM. The difference is that opposed to the keys being stored in a common HSM shared between clients, each client has a private HSM, which may be important to comply with certain regulations (Huang and Chen, 2018). Cloud HSM generally offers an HSM certification with a higher score in security. AWS (AWS, 2020a) and Google Cloud (Google, 2020a) have cloud HSM offers.

### 2.3.4 *Other management strategies*

Aside from the Hardware Security Modules, there are multiple services where the keys/secret information is stored in a centralized server owned by the company offering the service. The data that a user stores in those servers is protected by end-to-end encryption. This means that every piece of information on the servers is encrypted with a master key defined by the user, so that no one, except for the owner, can access it. This strategy is used by many password managers including Apple Keychain. This service uses AES-256bit encryption, performed with the user's master key (Apple, 2020). To increase the level of security, two-factor authentication is recommended, with biometric data or multiple device authentication when accessing the keys. This is used to prevent access if the key is compromised.

One of the conveniences offered is multi-device support. That consists in replicating the keys in every device of the user and synchronizing all the information with the cloud, to maintain consistency. This also acts as a backup to protect the user from undesired losses.

This method is very convenient and a good approach to store small amounts of data. It has very easy accessibility and is always available, but in the case of the Keychain, it has software integration limitations with non-Apple devices. Also, the servers where the data is stored are not certified like in a KMS, and although they are in the cloud and can be accessed anywhere, they do not offer the convenience of integration with a cloud provider.

### 2.3.5 *Shamir's Secret Sharing with multi-signature*

In all of the approaches mentioned above, there is a single master key that holds access to the keys stored in a KMS, HSM, Keychain, etc. Although it is also possible, with key hierarchy, to have different keys that have a different kinds of permissions, there is always the premise of having a master key owned by an individual or shared by a group. For a group of people, two options are: trust the master key to an elected individual or have the same key shared by multiple people. In the first scenario, there are multiple problems that can occur like corruption and abuse of power, or a tragic accident like death, that could lead to loss of access to the keys forever. In the second scenario, there is also the problem of abuse of power by one person of the group, that could perform actions with the keys without permission. Both scenarios represent real problems.

Shamir's secret sharing tries to solve these problems by offering a different approach to master key ownership. Instead of having a key shared by a group, this key is divided into different pieces of information that alone do not hold any value, as they individually are not enough to perform any action with it, as seen in (Steinfeld et al., 2007).

The only way to perform an action with the key is by combining this method with multi-signature. This happens when it is required for every holder of a part of the key to making a signature. This does not expose the master key or each of the secrets and the signature is only valid if everyone signs. There is also the option of agreeing beforehand that a certain percentage of the pieces are enough to perform actions, for example, 51% of the group. This solution is an effective way to solve the multiple ownership of secret keys.

### 2.3.6 *Summary*

There are different strategies that serve different purposes and use cases. Some are designed for quick access, others for great security, and others to be used inside entities in an untrusted environment between multiple individuals.

Personal use of hardware security modules is a secure but also technically complex method, as the user needs to set up and maintain it. KMS or CloudHSM provides a simplified way of using HSM. They provide the same functionalities for key storage, key creation, and encryption of data. Both scenarios require a master key that grants total access to the stored keys. Shamir Secret Sharing is the only mentioned strategy that provides a solution to multiple ownership of the master key, inside an untrusted environment.

---

## STATE OF THE ART

---

While the previous chapter presented an overview of the fundamental techniques and approaches for storing sensitive or secret information securely, this chapter focuses specifically on crypto wallets. Crypto wallets serve two main functions: keeping crypto assets safe by protecting them from theft and cyber-attacks and providing a user interface for interacting with blockchains. Also, in recent years, hundreds of new blockchains were created, which lead to an enormous increase of users that need a crypto wallet. A large percentage of these are non-technical users, which means the management of and interaction with crypto-assets should be as simple as possible, to fit this type of user. It is no longer only about securing crypto assets as secure as possible, it is also about securing them in an accessible and easy to operate manner.

A crypto wallet is secured by a cryptographic key, a private key, or by a 12 to 24 words mnemonic phrase. The user that has access to one of them holds the ownership of the wallet, so the security of the key is extremely important (Binance, 2020b).

Currently, there are several approaches to store private keys that hold assets (Jokic, 2019) and also to make them accessible to sign transactions with (Pal et al., 2019). Some put security at risk, and others are very hard to use for beginners, with the most used types of crypto wallets, as well as their advantages and disadvantages, presented in this chapter.

### 3.1 WEB WALLETS

#### 3.1.1 *Hosted*

Hosted web wallets are considered to be the most user-friendly type of crypto wallets. In this type of wallet, a user creates his account with a custom password that secures every asset held by the user. The process of storing the private key is completely hidden from the user who does not have any control over it. This key is stored by the web server itself in its database and the user needs to trust its funds to a third party, as shown in Figure 4 (Jokic, 2019). Some of these services use different techniques to securely store private keys, like hardware security modules.

Although this may be very convenient for the person using the platform, it can also be extremely dangerous, as the service can be the target of a major attack that could steal a large number of crypto assets (Guri, 2018). Database exploits and stolen credentials are the main attack vectors of this approach. There is also the disad-

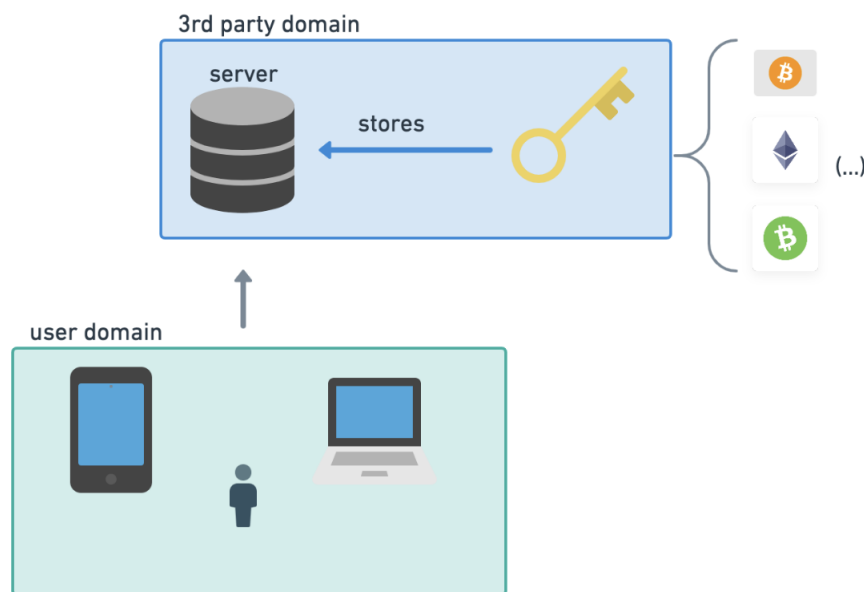


Figure 4: Hosted web wallets

vantage of the user not being in possession of the private key, as outside of the web application it cannot be proven that he is the real owner of the crypto assets. This strategy is not seen by the crypto community as being correct, as it goes against the crypto main motto: “Not your private keys, not your coins”. These wallets also have the advantage of being able to support multiple cryptocurrencies very easily and providing instant liquidity to make exchanges (Di et al., 2020).

From the numbers of the most popular hosted web wallets, these seem to be the most popular type of wallet, because these attract the users that want to create an account and start trading as easily as possible without having to deal with private key storage responsibility. Blockchain.com and Coinbase have reported having 65 (Blockchain.com, 2021) and 42 million (Coinbase, 2021) users on their platforms at the end of 2020.

### 3.1.2 Non-Hosted

Non-hosted wallets are also accessed through a browser. But, as portrayed in Figure 5 do not store the key in their servers. The private key is asked for every time the user wants to log in (Horizen, 2020). So, in this case, the user is the holder of the keys and it is his responsibility to properly store them. The private key of the user is stored in the browser’s cache, which can be an attack vector that can lead to stealing the wallet (ICOHolder, 2020) (Li et al., 2014).

Usually, these wallets are implemented as browser extensions, which improves the usability of decentralized web applications, as these make it easier to sign transactions and interact with the application without leaving the web page. As seen in Figure 6 (Metamask, 2020), a website presents the user with the details of the transaction

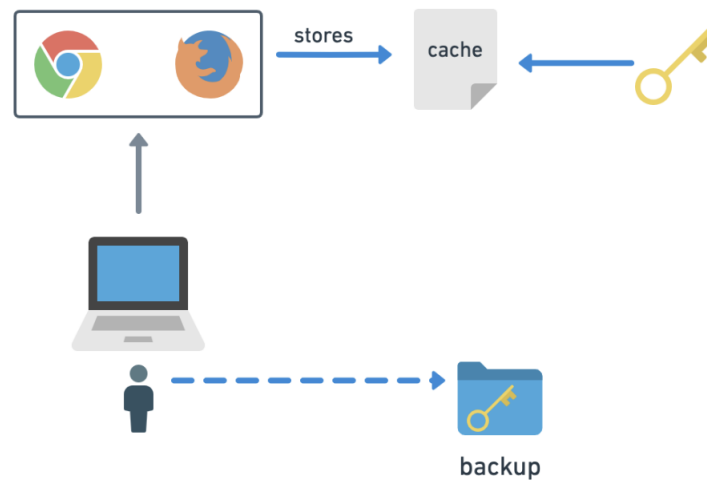


Figure 5: Non-hosted web wallets

to be signed and a button to use a wallet to sign the transaction. The wallet plugin allows the user to accept and sign the transaction without leaving the website. This is one of the biggest reasons for the success of non-hosted web wallets, like Metamask and MyEtherWallet, that have reported having more than 1 million active users (Knowles-Rivas, 2021) and are essential for the core functioning of many web applications. Asset trading and games are some of the most popular examples. These wallets played a very important role in blockchain web expansion by making these applications accessible to most users, with very simple interfaces and good usability.

### 3.1.3 Software wallets

Software wallets, be it desktop or mobile, store private keys in the user's devices (Figure 7).

Desktop wallets run offline. In this approach, the user is the holder of the private keys that are provided to him and exported to a file stored locally, which the owner should encrypt (Jokic, 2019). This may raise some concerns, because the file may be lost or damaged, which would result in not being able to access the wallet ever again. Some desktop wallets provide encryption natively, by using a password given by the user and transforming it into a 256 bit key for AES, TwoFish, etc.

It is crucial to have a backup of the key or seed phrase. Another concern that is often brought up is the fact that the computer, when online, has multiple attack vectors (e.g. phishing, malware, etc.) (He et al., 2020) that may result, once again, in losing control of the wallet. If the user is using the encryption provided by the application, his crypto wallet file might be in danger in the case a weak password or cipher is used (brute force). Alongside being the owner of the private keys, the other big advantage of this type of wallet is that if using an air-gapped



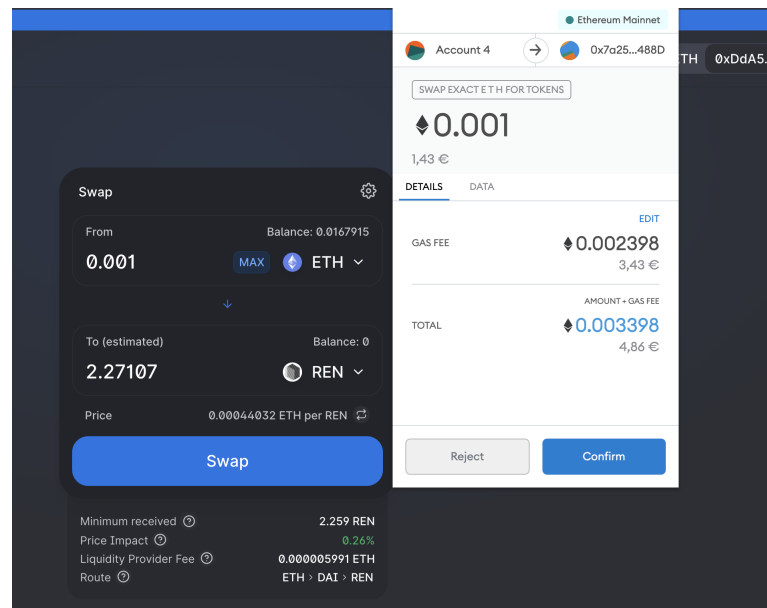


Figure 6: Plugin non-hosted web wallets

system, it can be a very secure system with few downsides, but less practical. Mobile wallets are very similar to desktop wallets, as these are also vulnerable to hacking and phishing attacks and carry also an increased risk of being stolen or lost, due to being mobile devices.

Mobile wallets may be the second most popular type of wallet in the market, with platforms like Trust Wallet, Coinomi, and Coinbase Wallet. Each of these apps has more than 1 million downloads on the application stores, but very far from the numbers presented above for web wallets. Desktop wallets are very difficult to evaluate, as the most popular platforms do not record the number of installations that are done, as many users use the wallets only offline. With that said, some popular wallets among the community are Electrum and Exodus.

### 3.2 HARDWARE WALLETS

Also known as cold storage wallets, these do not have an ongoing Internet connection and can be electronic devices or pieces of paper. Usually, the electronic devices are also protected by a pin number (Di et al., 2020) that offers an extra layer of security in the case of theft of the device (Bulut, 2019). To execute operations, the user needs to connect the device to a software wallet or to a non-hosted web wallet but the private keys never leave the device, meaning this is one of the most secure methods to storing private information. Some of these devices are reasonably expensive but are the most recommended way to store large amounts of cryptocurrencies. The biggest downside of these wallets is their lack of convenience and practicality.

The market of these wallets is dominated by Ledger and Trezor (Jokic, 2019). Ledger has reported in 2020 more than 1 million devices sold around the world (Ledger, 2021). That number may seem small compared to

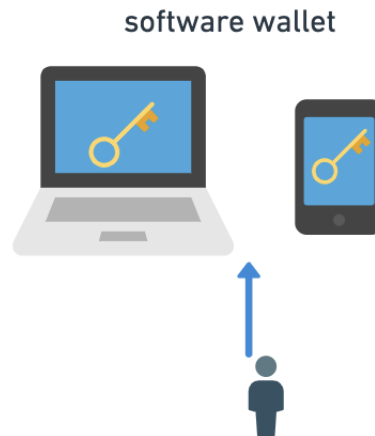


Figure 7: Software wallets

web wallets, but usually, these are very technical users that store large amounts of funds. As for Trezor, there are no exact numbers made public.

Paper wallets (Figure 9) are just a piece of paper with the private key written or a QR code that can be scanned. Because there is also the need to use a software wallet, after the paper is scanned, the hardware security properties are lost. This method is discouraged because the wallet can be easily destroyed and does not have the same security properties as the electronic cold storage devices. Both approaches (paper or electronic) are not well-suited for non-technical users and are considerably less practical than wallets connected to the web, especially when making regular transactions. Additionally, electronic hardware wallets are considerably expensive.

The paper wallet market is mainly occupied by BitcoinPaper Wallet in the bitcoin community, and many other websites that generate the wallets for every different cryptocurrency. It is difficult to speculate on how many users use this approach to store their private keys, as any person can create his paper wallet without needing any service.

### 3.3 MULTISIG WITH MULTIPLE DEVICES

A multi-signature (multisig) approach can be applied to every type of wallet already presented. If a wallet is seen as a vault, the typical single signature model could be seen as a single padlock that is protecting the safe. The person that has the key to that padlock, is the rightful owner of the treasure inside. If a multisig system is considered, the vault would have two (or more) locks, and the treasure would only be redeemed if every key holder unlocks its padlock. A single key is not enough to open the vault, as represented in Figure 10.

If a user does not need any resource other than the key itself, it may be dangerous, especially to less experienced users that may not have the knowledge on how to properly store their private keys (Di et al., 2020).

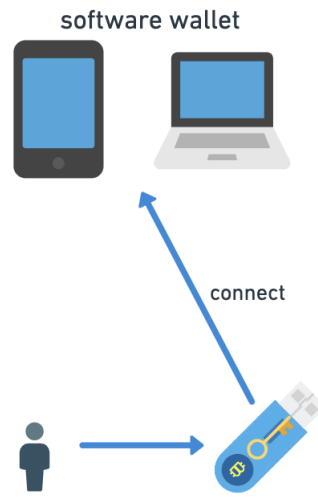


Figure 8: Hardware wallets

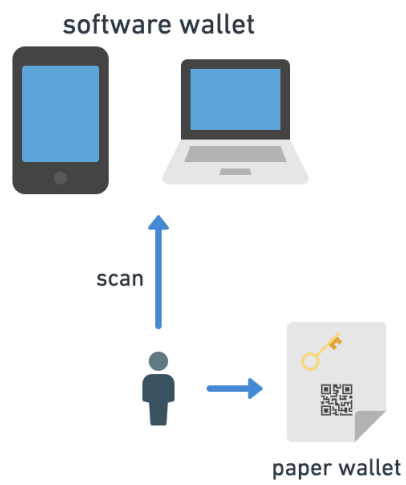


Figure 9: Paper wallets

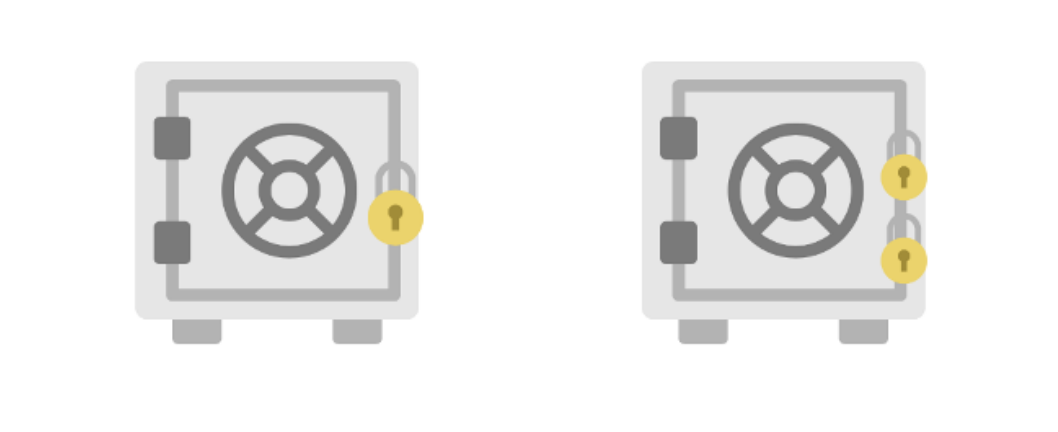


Figure 10: Multi signature wallets representation

Another use case, if a wallet is owned by more than one person (e.g. a corporation), the key would have to be entrusted either to one single individual or shared by multiple people, which could lead to individuals being able to make unauthorized transactions (Zhu et al., 2017). Multisig solves this, as the assets held by the private key would be associated with a multisig address and require multiple signatures to be accessed or used (Di et al., 2020) (Binance, 2020a). If the holders of the funds need to make a transaction, they would need every address (or some of them) to sign it using their respective private keys, solving the multi-owned wallet problem.

The multi-signature could be done in different combinations as in Figure 11, for example, the wallet could have three holders and only require two of those people to sign the transaction. This can be done with multiple variations, requiring the majority of the holders to be enough to sign a transaction or require every holder to sign it.

This method could also benefit mobile or desktop wallets because if a device gets compromised by a phishing attack, it no longer puts every asset held in the wallet at risk. Imagine that a user named Alice creates a multisig address with three holders that requires two signatures for a transaction to be approved, as seen in Figure 11. Alice could save the three private keys in different devices, like using a hardware wallet to hold one of the keys, a mobile wallet to hold the second one and a desktop wallet to hold the third.

If her mobile phone is stolen or hacked, the wallet is no longer compromised, because she can still access her funds by using the other two wallets and the hacker/robber is not able to sign transactions, because two keys are needed. The same happens if she lost the hardware wallet, as the assets would remain in her possession. This is one of the major advantages of multisig wallets as the increase of security is enormous. This scenario is represented in Figure 12 As a downside, it has the difficulties of setting up a system like this, as it may be very technical and not accessible to any user.

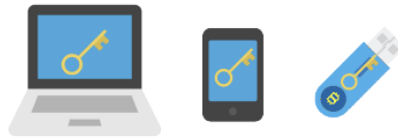


Figure 11: Multi signature wallet with multi devices

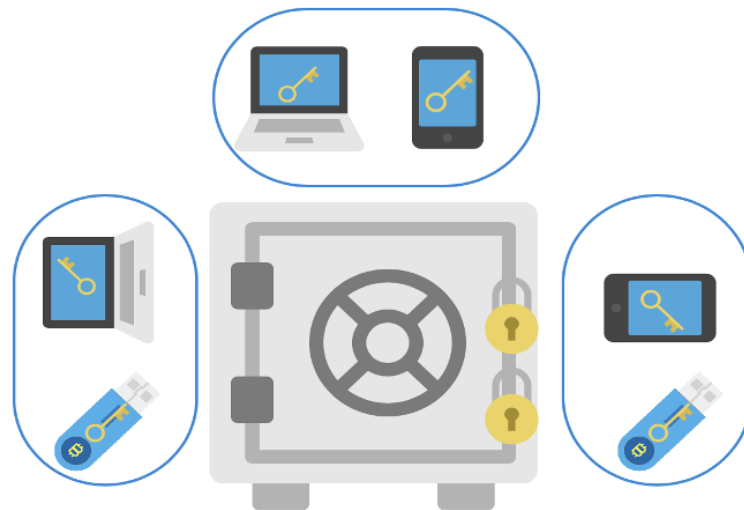


Figure 12: Multi signature wallet with multi devices unlocking

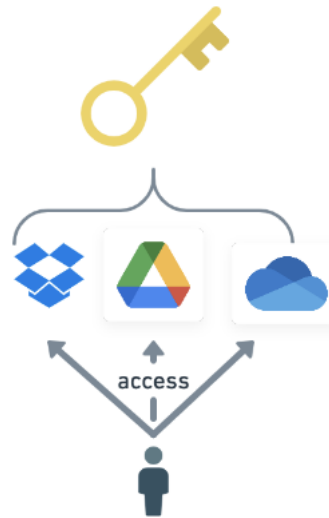


Figure 13: Multisig wallet with multiple cloud storage providers

### 3.4 MULTI-CLOUD WALLETS

Multi-Cloud wallets merge two concepts in a single product: multi-signature and cloud storage. Similar to the multisig wallets approaches shown above, these wallets separate the private key of the user into different pieces. But instead of storing them in different devices, they are stored in multiple cloud services that the user already uses, as shown in Figure 13, with Dropbox, Google, and OneDrive as an example of providers. To sign a transaction and have access to the key, the user needs to access all the cloud services where parts of the key are stored. If one of them is compromised, the wallet remains safe, as one part of the key is not enough to cause any damage. Another difference is that multi-cloud is, at least in its simpler form, targeted at single-user wallets. It is, however, assumed that the cloud providers do not collude to access the user's funds/assets.

With a traditional multisig wallet, that uses different devices, all of these are vulnerable to loss, theft, or hardware failure, which can lead to total key loss, whether these are paper, hardware, or software wallets. With multi-cloud, the key is safe from this type of vulnerability, as losing the key would require the user to lose access to a subset of the providers or these to fail. This approach also takes advantage of the fault tolerance mechanisms employed by cloud providers.

As for usability, multi-cloud also offers global accessibility, because the key is in the cloud. But this approach has the significant advantage of the user is in total control of every piece of the key, as it is not stored in a centralized server with no access from the user. Multi-cloud wallets are a very recent concept and, to the best of our knowledge, the only wallet in the market is Keyruptive (Keyruptive, 2021) that has a patented technology on this approach and is involved in this project.

### 3.5 DISCUSSION

The usability aspect of a wallet is as important as its security, to make it compelling for the average user while offering minimal risk to his assets. Every wallet approach mentioned before has some downsides.

The web wallets (hot wallets) privilege the usability simplicity, while the hardware wallets (cold wallets) focus more on security against account or credentials theft. It was decided that the main focus of this work would be on using multi-cloud technology, as it privileges security with a good balance between theft vulnerability and stolen credentials, and improve its usability, by having a similar interaction to a web wallet. As mentioned, there are, to our knowledge, very few applications of multi-cloud technology in this market, and this work is intended to explore multiple variations of it.

Part II

CORE OF THE DISSERTATION



---

## APPROACH

---

The cryptocurrencies' market has been growing and there are now millions of users holding crypto assets, with some of these having market caps valued in billions of US dollars. This growth affects the crypto wallets' market, as with the increasing number of blockchains, cryptocurrencies, and other crypto assets, and their valuator, the need for increasingly secure yet usable wallets is highlighted. Particularly, with the increase of non-technical users, the market needs to adjust and offer a better user experience without sacrificing security, which remains one of the most important aspects of a wallet. The growth of web decentralized applications opened a path to a new type of wallets that are implemented as browser plugins. These are important to improve the experience of interacting with these web apps. Some plugin wallets are very successful products in the market with a large number of users.

The main goal of this dissertation is to combine different positive aspects of different wallet implementations in a single product, able to provide good security, usability, and web app interaction. Multi-cloud technology will be the basis of this implementation, which will be objectively compared, using known metrics, to established wallet approaches.

In this chapter, we discuss the system architecture of different approaches. In subsequent chapters, we describe the implementation of a set of representative wallet mechanisms following these system architectures and then compare them from the point of view of usability and security.

### 4.1 MULTI-CLOUD APPROACHES

In our multi-cloud approaches, the private key is ciphered with an encryption key and then divided into three parts, each of these stored in a different cloud provider in a specific file. One of the cloud providers has a file with the corresponding public key of the wallet. This is necessary to import the account to the web non-hosted wallet without having to decipher and access the private key. The public key does not need to be separated or ciphered because it can only be used to see the public information of the wallet.

There are different techniques to store the encryption key: it may be stored on a server, secure enclave, password manager, or others. In this section, we will explore these three techniques. They have different vulnerabilities and usability, so a final discussion is necessary to compare the advantages and disadvantages of being used in this context, as well as a comparison to the already existing approaches in the market.

#### 4.1.1 *Server authentication*

In this approach, the encryption key is stored in an external server, as seen in Figure 14. In the setup process, the user creates an email and a four-digit pin code that are used as authentication to the server. The encryption key is ciphered with the chosen pin code, so it is not stored in plain text. This means that the administrator of the server has no access to the encryption key.

When signing a transaction, the email and pin code are requested after authenticating to the three cloud providers. The ciphered encryption key stored on the server is deciphered with the pin code requested. Then, the ciphered private key is deciphered with the mentioned encryption key. If this results in a valid private key, it means that the pin code was correct and the transaction can be signed. To prevent brute force attacks, the server locks after three consecutive failed attempts for a determined amount of time. On the sign up process, the user is given a twelve-word backup phrase, that should be kept safe at all times. This phrase is able to restore the wallet, which could be used if access to the wallet is lost or compromised.

In terms of usability, the main advantage is being compatible with any device and accessible from anywhere, as everything is stored in the cloud. Another key aspect is that the user has control of the private key and the administrator of the server has no access to it. If the server is compromised or fails, there is no loss for the user, because the wallet can be restored with a recovery backup phrase that is given to the user in the initial setup process. Even if the three cloud providers conspired, it would not be enough to expose the key, as it is ciphered.

As for security, this approach may be vulnerable if the user's device is compromised with a key logger, which may lead to stealing every password. This could lead to a total loss of the funds.

#### 4.1.2 *Secure enclave*

This approach takes advantage of the hardware secure enclave that is available in certain devices, that have a chip that stores cryptographic keys that can be used to cipher and decipher data without being exposed, as this process is executed inside the chip. Ideally, a key from the chip could be used as the private key and the signature process would take place inside the secure enclave similar to hardware wallets. Due to format and size incompatibilities with the keys used in popular blockchains, this is not possible, so this method had to be adapted. In the proposed implementation, a key from the secure enclave is used to cipher the private key and is accessed with the device's enclave authentication. To sign a transaction, the user authenticates in the three cloud providers to access the ciphered private key and then log in with the secure enclave credentials. If successful, the private key is deciphered and the transaction is signed. There is also a backup seed phrase that can be used to restore the private key, identically to the external server approach.

As for usability, this technique can only be used in specific devices with the mentioned hardware. Also, as the encryption key is stored in a specific device, the user can only use that device to sign transactions. It requires an account to give permission to the secure enclave, which may be different depending on the device. A big

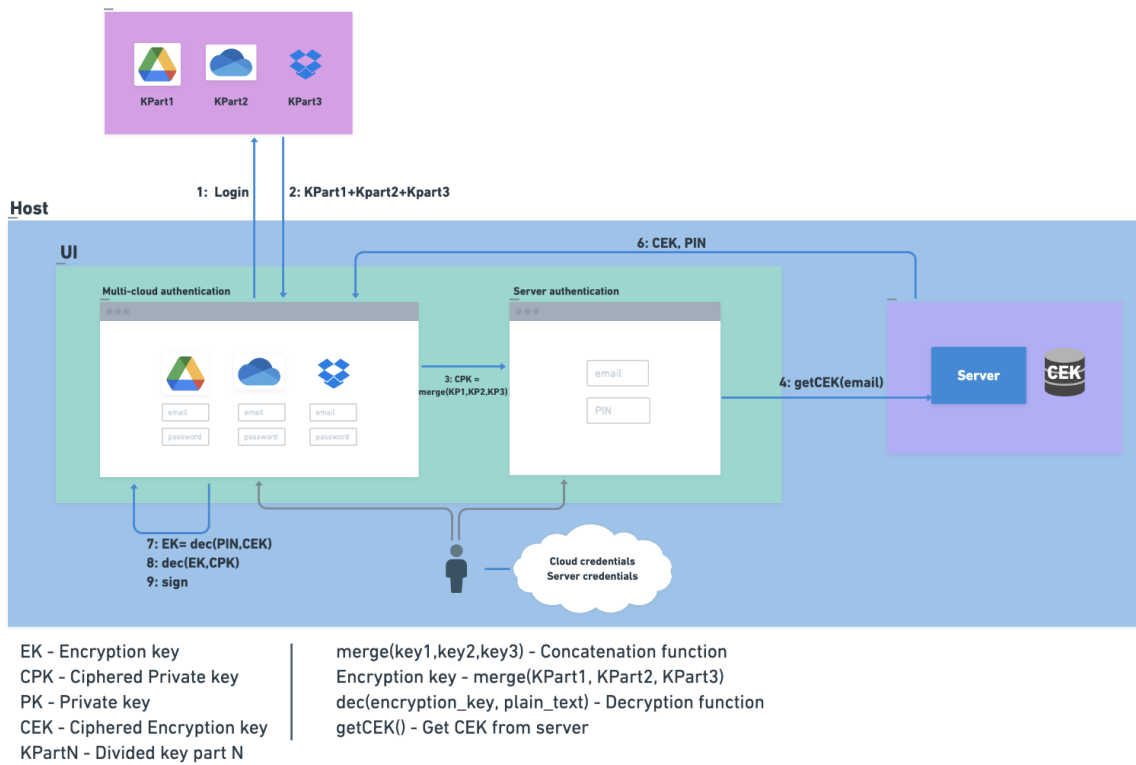


Figure 14: Server authentication architecture

advantage of this approach is that the private key is in total control of the user, that does not depend on any external server.

In terms of security, the main advantage is that the encryption key is never exposed and never leaves the user's device. Even if the cloud providers were compromised, an attacker would need access to the machine itself and have the authentication credentials to access the secure enclave and decipher the private key. The main disadvantage is that loss, theft, or hardware failures may imply loss of access to the funds if the backup seed phrase is not stored. In the case of a compromised device, the attacker may gain permission to access the enclave and the clouds with a key logger, which would cause a total loss of the funds.

#### 4.1.3 Password manager

In this approach, the encryption key is stored in a password manager that is accessible with an account. The password manager stores the key encrypted by a master password in a database. In order to sign a transaction and decipher the private key, the user needs to authenticate with the password manager credentials. The encryption key is read, the private key is deciphered and finally, the transaction is signed. As in the other proposed implementations, there is a backup seed phrase to restore the wallet, that the user should store safely.

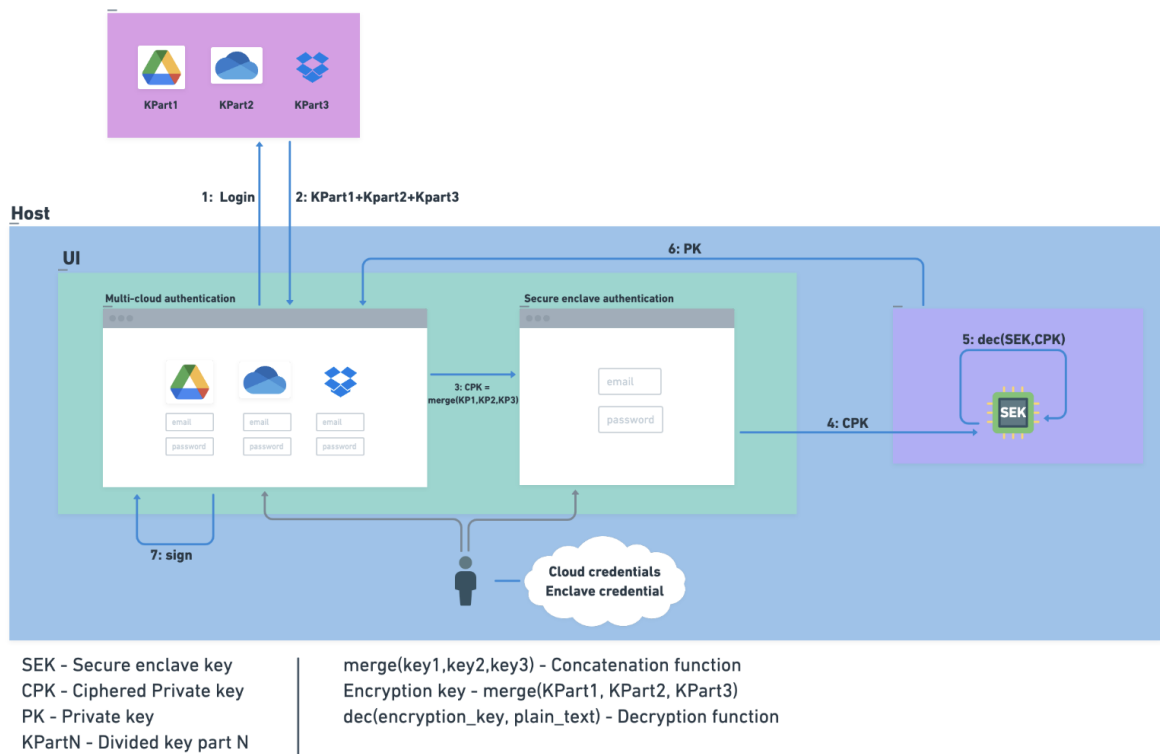


Figure 15: Secure enclave architecture

The main advantage in usability is that a password manager works between every device owned by the user, as the keys are stored in the cloud. The private key can only be accessed by an attacker that has access to the password manager account and the various cloud providers. If the device is compromised, the credentials may be stolen with a key logger.

## 4.2 OTHER APPROACHES

For completeness, it is important to study other approaches for key storage that should be compared with the proposed ones in functionality, usability, and security.

### 4.2.1 Centralized Server

In the centralized server approach, the private key is stored in an external server. The user does not own the private key, as does not have access to it. Every transaction ordered by the user is signed in the server, which the user accesses with an authentication login.

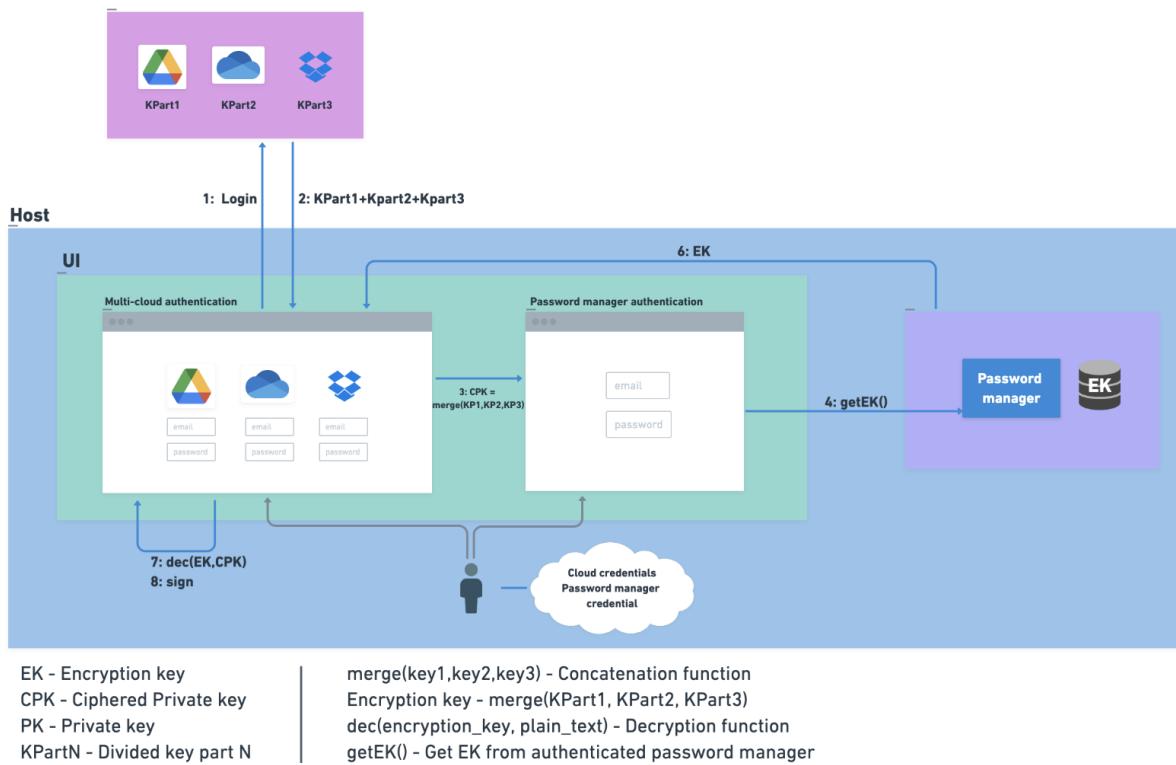


Figure 16: Password manager architecture

The biggest disadvantage is that the user does not have ownership of the private key and has to trust a third entity that has the ability to control the wallet. Also, the server may be the target of an attack or have access failures. If it is compromised, the user may lose access to the wallet or lose the funds completely.

#### 4.2.2 Web local storage

This approach stores the ciphered private key in the browser's local storage. It is ciphered with a password provided by the user, that is used to unlock the account when starting the session.

After introducing the password, the private key is unlocked and stored in memory. This may be an attack vector that can expose the private key. Losing the device may lead to losing access to the wallet if the user did not store the backup seed phrase that can restore the wallet. The main advantages are in usability, as the user has access to the private key and the wallet only requires one password.

#### 4.2.3 Hardware wallet

The user needs to have an external device connected to the computer and unlock it. For this, the user inserts an eight-digit numeric pin code. To sign a transaction, the user has to confirm it on the web interface and then

proceed to confirm it on the external device. Here, the user can see the various parameters of the transaction and accept it. The status of the transaction is presented on the web interface on the computer screen.

In terms of usability, this approach requires having a specific external device. As for security, the private key is never exposed and never leaves the device. However, losing it implies loss of access to the wallet, unless a recovery backup phrase is used.

### 4.3 PROPOSED ARCHITECTURE

The proposed architecture has two parts: a non-hosted plugin wallet and a multi-cloud connection platform. The first is an existing wallet that allows a user to sign transactions using an imported account. This should be compatible and allow interaction with blockchain applications. The second is the platform accessed in the browser where the user will be able to connect to the three cloud providers and sign transactions. The wallet communicates with the multi-cloud platform via Window Web API, which allows communication between opened browser windows without the need for an external server. The functionalities to implement are: importing accounts and signing transactions, as seen in Figure 17.

When importing an account, the user should authenticate to the cloud service that has the public keys file, containing multiple public keys. Then this file is read and the array of public keys is sent as a message to the non-hosted wallet. The wallet allows the user to import one of the accounts.

When signing a transaction, the connection platform receives an unsigned transaction. The user should authenticate to every cloud service. The ciphered private key is read and then depending on the storage approach, the various steps to decipher it are presented to the user. In the case of the external server approach, the user is asked to introduce his email and pin code. In the secure enclave and password manager approaches, the user has to introduce the email and password to unlock the account. After having the private key in plain text, the transaction is signed and sent to the wallet, which displays the signed transaction. The private key should be erased from memory after this process.

The approaches that will be considered are the proposed multi-cloud approaches, that will be implemented as part of this work, the hardware wallet, and the web local storage that is already implemented as established products.

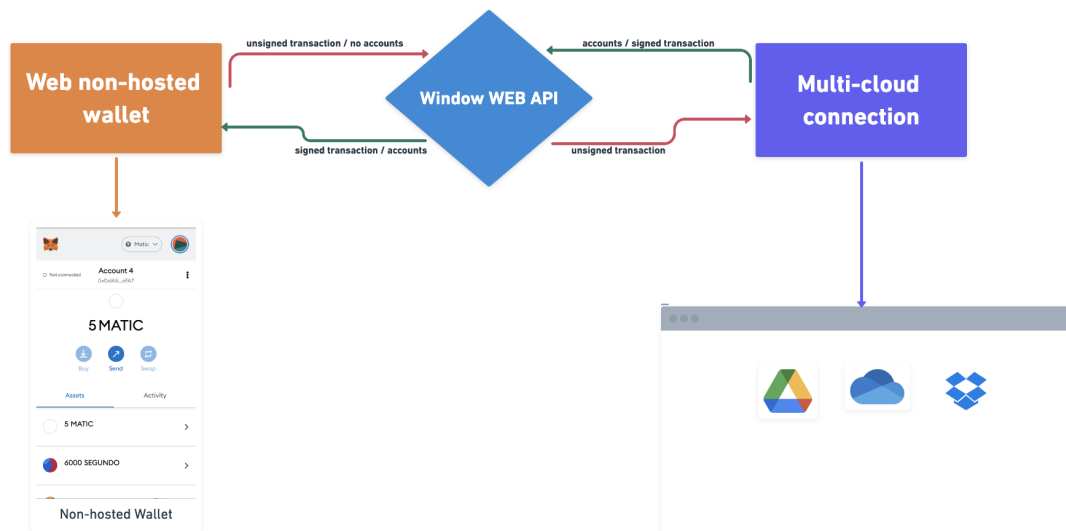


Figure 17: Architecture

---

## IMPLEMENTATION

---

A possible approach for the browser plugin would be to build on Metamask, that is a popular non-hosted wallet of the Ethereum blockchain, that supports decentralized web applications. Metamask is open-source, updated very frequently, and maintained by a team of developers, so it is already a very mature product (Choi and Kim, 2019). As for the multi-cloud wallet system, the goal is to leverage Keyruptive's technology, building on the technology (Pontes et al., 2017).

As mentioned in the last chapter, multi-cloud with the server or with password manager are what we found to be the best approaches in terms of usability and security.

### 5.1 IMPLEMENTATION STRATEGY

The first step was to do an extensive study of the Metamask open-source repository (MetaMask). To find a way to import accounts and send transactions from an external source where the multi-cloud technology would be applied.

Metamask has a feature that allows a user to connect an external hardware wallet with its interface. This makes importing accounts and signing transactions with the hardware device connected and unlocked possible. When importing an account, the addresses stored on the device are presented in the Metamask interface, where the user can choose to import one. When signing a transaction with one of these accounts, the transaction is sent to the device and signed inside it upon confirmation by the user. The final signed transaction is sent to Metamask and shown in the interface.

These are the functionalities that to implement, so, we focused on the hardware wallet connection and interaction part of the code, with the intention of creating a multi-cloud connection platform that communicates with Metamask as if it was a hardware wallet. Similarly to a hardware wallet, the signing of a transaction would happen outside Metamask, in the multi-cloud platform.

To connect different hardware wallets from different manufacturers, the Keyring class acts as an interface to communicate between the wallet and the brand's device. Each brand has its own implementation of this class to meet its specific functionalities. The main methods needed to import accounts and sign transactions are: `getAccounts()` which returns an array of addresses, and `signTx()`, that returns a signed transaction. The two currently supported brands are Ledger and Trezor. With Ledger, the user interacts with the screen of the hardware device. With Trezor, a window popup is opened inside the browser where the user can connect the



wallet and perform various actions. In order to make our version of the class, we decided to use the same strategy as the Trezor Keyring, which would also have a popup opened where the user could log in to the cloud services, import an account and sign a transaction. The result of these actions would be communicated to Metamask, using the Window Browser WEB api.

## 5.2 WALLET CODE ARCHITECTURE

Figure 18 presents the code architecture with our version of the Keyring class and the KeyruptiveConnect class, that communicates with the external platform with which the user interacts. It has two main methods: `getAccountsCloud()` that opens the popup and creates a listener that expects to receive an array of addresses; and `signTxCloud()` that starts by opening the popup with a message containing the unsigned transaction in hexadecimal format and then, creates a listener expecting a correctly signed transaction.

The Keyring class calls these methods from its `getAccounts()` and `signTx()` methods, respectively. While it is expecting a response from the KeyruptiveConnect class, it puts the Metamask wallet in a waiting state, that is interrupted when the popup window is closed. In case the user closes the window before the various processes are complete, an error message is presented on Metamask.

## 5.3 MULTI-CLOUD CONNECT PLATFORM

The connection platform is a web user interface. It has two pages: to import an account and to sign a transaction. Both allow the user to log in to the necessary cloud services. In each of them, a part of the user's ciphered private key is stored, in the file called `private.json`. To join the three parts of the ciphered key, the user needs to log in to the services, which requires one click if the authentication is saved on the device. If not, the user needs to insert his credentials with the possibility of using a password manager to simplify the process. The public keys are stored in the file called `public.json` in plain text.

Figure 19 presents the process of importing an account. In step one, the user has to connect a new hardware wallet and choose the multi-cloud option, that is the selected one of the figure. In step two, the user has to authenticate to the cloud provider that has the `public.json` file. In this simplified implementation, there is only one cloud provider. The file is fetched, read and, if valid, the popup is closed. In Metamask, the public keys are converted to addresses and shown in the interface, where the user can choose one to be imported, in step three. Finally, the new imported account is shown on the main menu, as in step four.

Figure 20 presents the process of signing a transaction. Firstly, the user has to create a transaction on Metamask and confirm it. Secondly, the multi-cloud connection website is opened and the user authenticates to the cloud providers. After this, a button will appear on the screen to sign the transaction, as well as the details of it (e.g. gas price, gas limit, etc.). After confirming this step, the files containing parts of the key are read and then, the user has to go through an authentication process to decipher the private key and sign the transaction. This process is different depending on the multi-cloud approach, detailed in the following sections. Finally, on

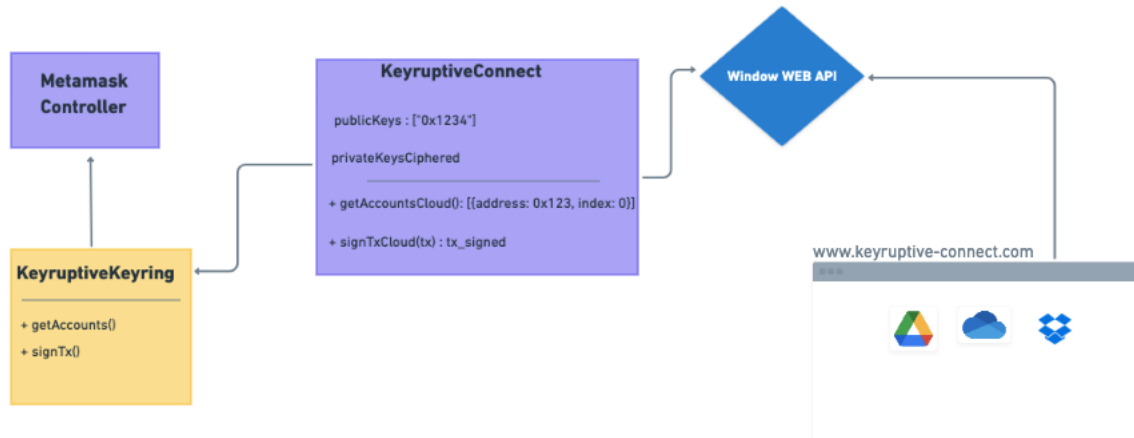


Figure 18: Code architecture

step number three, the multi-cloud website is closed and the confirmation of the transaction is presented on the Metamask wallet. In every approach, the private key is erased from memory by changing its value to null.

### 5.3.1 Multi-cloud with server approach

Figure 21 shows the sequence diagram for this approach. A form requesting an email and a pin code is shown to the user. The ciphared encryption key stored on the server is deciphered with the requested pin code. If it is correct, the ciphared private key is deciphered. Finally, the transaction is signed and the keys are erased from the memory of the browser.

The platform communicates with a REST API connected to a database that stores the email and ciphared encryption key of each user. To prevent multiple tries, it is also stores the number of failed attempts and a timestamp. If there are three consecutive failed attempts, then the timestamp is advances to a few hours in the future. When the user is authenticating, this timestamp is checked to see if the user is in a lock period. This makes the approach less vulnerable to brute force attacks.

### 5.3.2 Multi-cloud with password manager approach

In this implementation the Apple Keychain is used as the password manager. Figure 22 shows the sequence diagram for this approach. It is similar to the server approach, as the user is requested an email and password

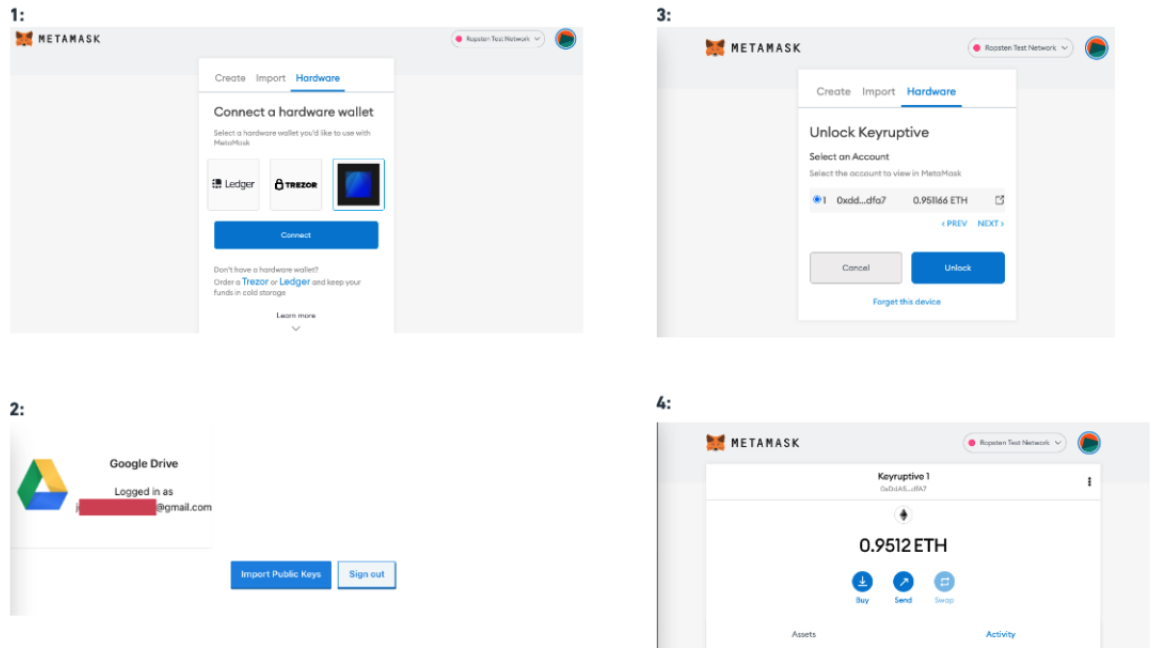


Figure 19: Import account from multi-cloud platform

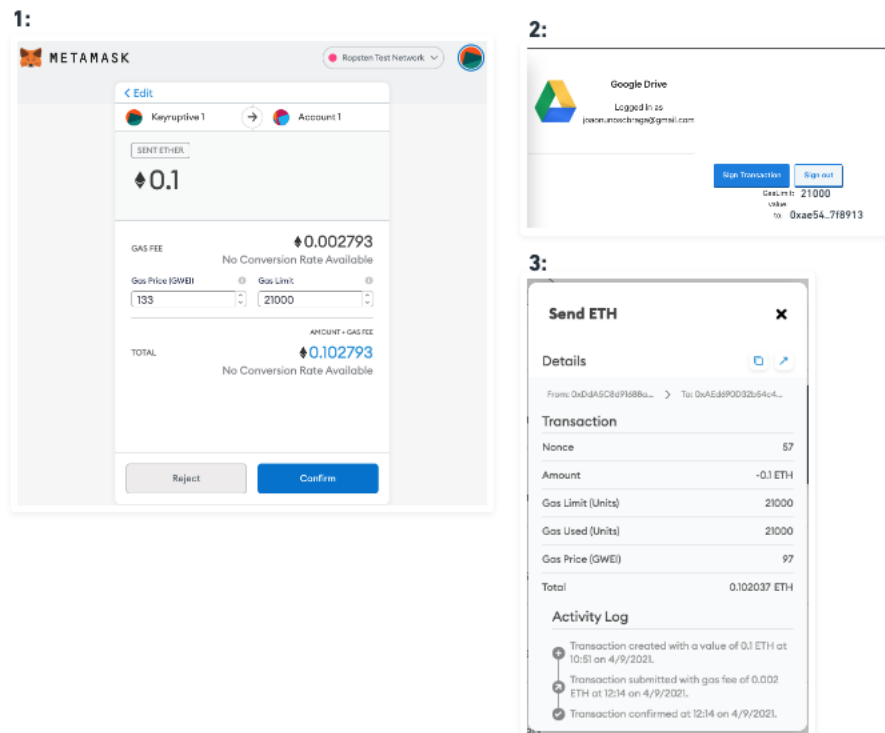


Figure 20: Sign transaction on multi-cloud platform

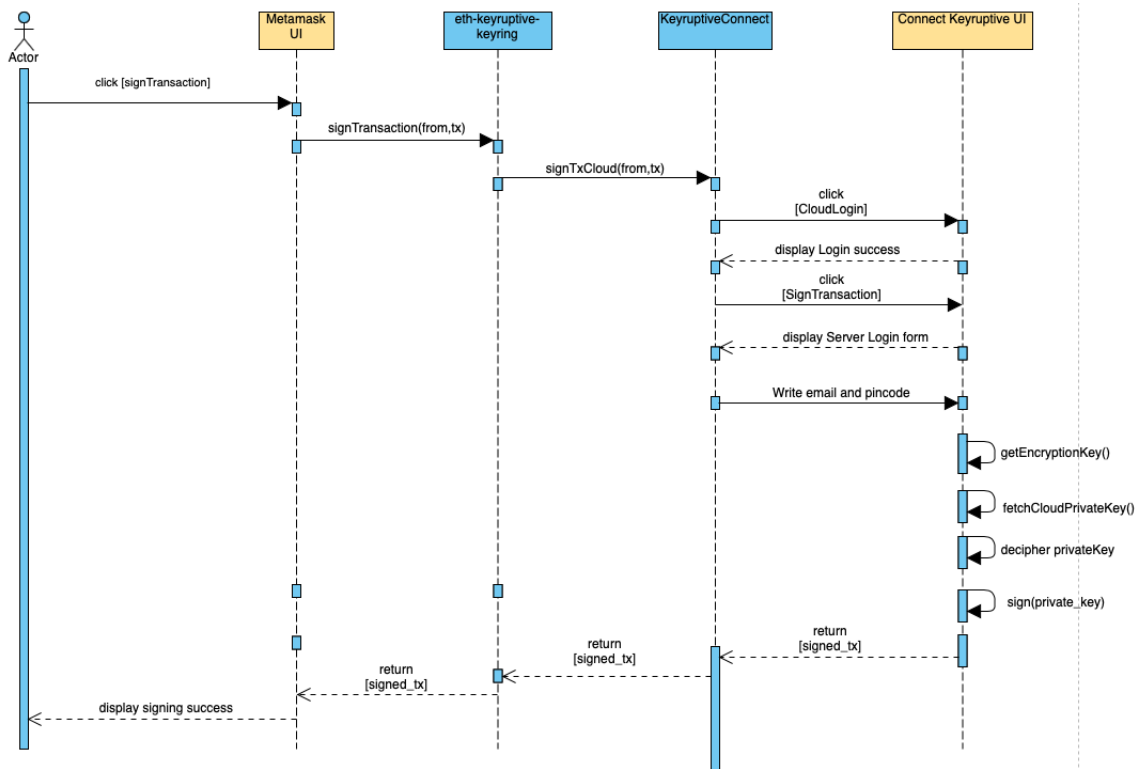


Figure 21: Sequence diagram for the multi-cloud with server approach

to access the password manager. After that, the encryption key is fetched, the private key is deciphered and the transaction is signed. A local server serves as an intermediary between the web application and the Apple Keychain. It is needed due to the fact that there is not, to our knowledge, an API to communicate directly between the Apple Keychain and the web app.

### 5.3.3 Multi-cloud with secure enclave

The sequence diagram of an implementation of the multi-cloud with secure enclave approach is presented on Figure 23. It is used the Apple Macbook Pro secure enclave that is available in the models that have a touch ID sensor. Due to technological constraints related to the current available APIs, this implementation had to use a separate desktop application that communicates with the secure enclave and the browser through an API.

For the signature process, the user has to authenticate with the user OS credentials. The private key is deciphered with a specific key from the secure enclave and the transaction is signed.

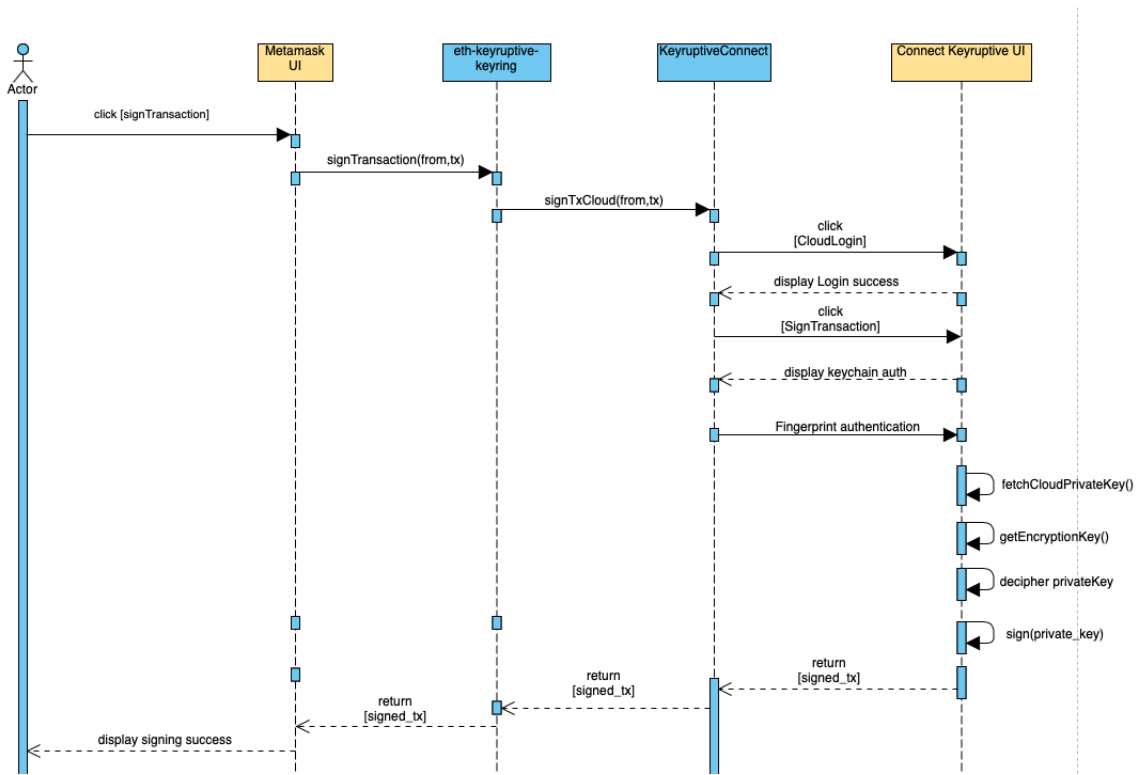


Figure 22: Sequence diagram for the multi-cloud with password manager approach

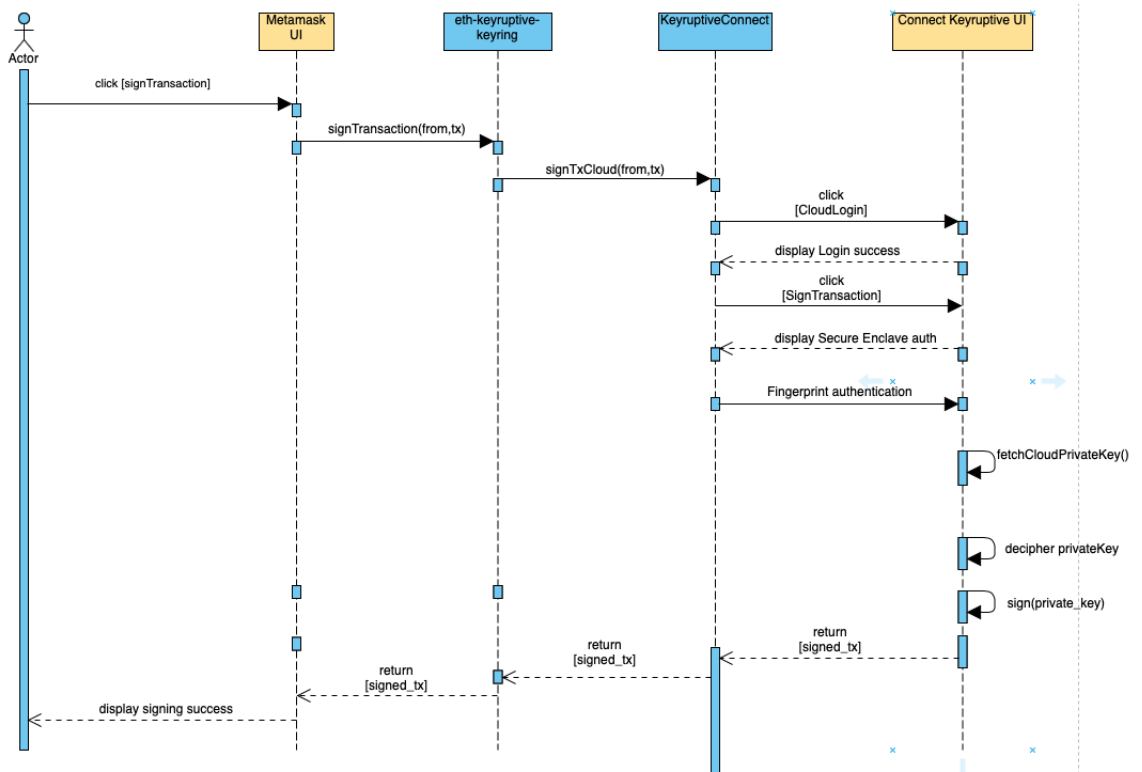


Figure 23: Sequence diagram for the multi-cloud with secure enclave approach

#### 5.3.4 Conclusion

The implementation of the three multi-cloud approaches was needed to have their functioning prototypes. These implementations, as far as our knowledge, are unique in the market and so, are essential to make a comparison between the various crypto wallet approaches. This will help achieve the final goal and understand from the comparison which crypto wallet performs the best in the proposed use case.

---

## EVALUATION

---

In this section, the multi-cloud wallet proposed in chapter 4 and its implementation, described in chapter 5, are evaluated regarding security and usability. For comparison, web local storage and hardware wallet implementations with the traditional Metamask and the Ledger hardware wallet.

The proposed implementations are objectively evaluated as follows:

- Usability
  - Signing transactions by how long and how many clicks it takes;
  - Managing multiple accounts and passwords using previous work ([Walia et al., 2020](#)).
- Security
  - How wallets compare regarding a set of attack vectors;
  - Vulnerability to common security testing tools.

### 6.1 USABILITY

Approaches use a different number of credentials, passwords and have different device compatibility. As seen in Figure 24, every multi-cloud approach requires multiple accounts and passwords, with one of these being a four-digit pin code on the multi-cloud with server approach. A larger number of accounts may be considered a disadvantage because it brings effort to the user. However, when using a personal device or a password manager, it is not as prevalent. The other approaches only require one password or pin code.

The multi-cloud with password manager and external server approaches have the advantage of being compatible with any device and being synced to the cloud, which is not the case for the hardware wallet, secure enclave, or web local storage. So, they can be accessed anywhere with any device.

Approaches	Number of login credentials	Number of passwords	Device compatibility	Private Key Ownership
Multi-cloud w/ Server	4	3 passwords + 1 pin code	Any device	yes
Multi-cloud w/ Password manager	4	4 passwords	Any device	yes
Multi-cloud w/ Secure enclave	4	4 passwords	Device specific	yes
Web Local Storage	0	1 password	Device specific	yes
Hardware Wallet	0	1 pin code	Device specific	yes

Figure 24: Credentials, device compatibility and private key ownership for different approaches

### 6.1.1 Number of clicks

The number of clicks is a way of quantifying the time and effort required to sign a transaction. The more clicks, mean user effort, as seen in (Poon et al., 2016).

As seen in Figure 25, every multi-cloud approach requires a similar number of clicks. These have an optimal effort when the user is using a personal device or a password manager because only one click is required to log in to each cloud provider. The multi-cloud with server approach requires slightly less input from the keyboard, as uses a pin code instead of a password.

In the hardware wallet approach, the pin code is typed in the external device which has only two buttons. Although it only requires one pin code, it needs a greater amount of clicks which translates to more time. In the others, the passwords are introduced on a standard keyboard. Traditional Metamask requires the least number of clicks, as it only requires one click after unlocking the wallet by inserting the password in a standard keyboard.

An attacker with access to the user's device, can use a key logger and steal every account, password, or access local files. This would imply a total loss of the funds.

### 6.1.2 Credential management effort

It is important to evaluate the compromises that the user has to endure in order to use the proposed approaches. The increased number of accounts and passwords may be an obstacle to usability, that may be prevalent in multi-cloud. It is important to consider the usage of an online password manager, that according to the work in (Karole et al., 2010), are the best password managers in usability. We want to compare objectively



Approaches	Worst case	Best case
Multi-cloud w/ Server	9 clicks + 4 emails + 3 passwords + 1 pin code	6 clicks + 1 email + 1 pin code
Multi-cloud w/ Password manager	9 clicks + 4 emails + 4 passwords	6 clicks + 1 email + 1 password
Multi-cloud w/ Secure enclave	9 clicks + 4 emails + 4 passwords	6 clicks + 1 email + 1 password
Web Local Storage	1 click + 1 password	1 click + 1 password
Hardware Wallet	55 clicks	15 clicks

Figure 25: Number of clicks and keyboard input

the usability of our approaches and the traditional ones with and without a password manager. For this, we have created a usability metric that evaluates the effort of memorizing multiple passwords and emails. This was based on the work of (Walia et al., 2020), which attributes a score to the strength of a password, which is measured by the number of characters, special characters and correlates it with how pronounceable the password is. In the presented context, a word is easier to memorize as it is more pronounceable.

It is shown in the study that passwords with less than eight characters have a weak grade and between eight and fourteen have an average grade. The latter is the most common password strength between users, as stated in (Walia et al., 2020) and has an average pronounceability of 60%. So, this is the password standard we are going to use. In the studied approaches, there are three types of passwords: 4 digit pin, 8 digit pin, and a password with the mentioned standard. In this context, emails are considered to have a high pronounceability and more than 14 characters.

The Figure 26 (lower is better) is based on the length of each string and how pronounceable it is. Strings under 8 characters are weak and their pronounceability is contemptuous. From this score and Figure 24, we can infer the score in Figure 27. The password manager scenario is included, as it seems to be a good combination with the multi-cloud approaches.

We can infer that without a password manager, the multi-cloud approaches have a noticeably higher score than the other approaches. In this context, this means that they compromise usability and convenience the most when compared to the others. When combined with a password manager, the credential management effort is much lower. The centralized server option scores the lower, as the user does not have to memorize any email or

Score		Pronounceability			
		0-25%	25-50%	50-75%	75-100%
Characters	<6 digits	1	1	1	1
	6-8 digits	2	2	2	2
	8-14 digits	6	5	4	3
	>14 digits	7	6	5	4

Figure 26: Combined effect of password length and pronounceability on the credential management effort (lower is better in usability)

password. The hardware wallet comes in second and takes no advantage of using a password manager. As a third, we have multi-cloud with server and web local storage with the same score. In the first one, the user has to introduce the server authentication email and pin code. The second one does not take advantage of a password manager, as the user always has to introduce the plugin password to unlock the account.

In conclusion, multi-cloud approaches' usability improves substantially with the usage of a password manager.

### 6.1.3 Discussion

In order to create a final comparison that combines the two explored methods and given there is not, to our knowledge, any literature combining these evaluations, it was decided to build a method to compare them. It was decided to add the amount of clicks (Figure 25) with the credential management effort (Figure 26). In this case, obtaining a lower final score is considered positive, because it represents a low effort in time and not requiring many accounts or passwords.

Once again, we have calculated the final score for two scenarios: with and without a password manager. In the case of the hardware wallet, it is considered the middle case on the number of clicks, that is the average between best and worst.

This final result is presented in Figure 28. It is possible to infer from it that the web local storage approach has the better score, mainly due to its simplicity of only requiring one click to sign a transaction. Followed by the multi-cloud with server and the other multi-cloud approaches. Once again, the hardware wallet approach is last, because of its large number of clicks.

Approaches	Without password manager	With password manager
Multi-cloud w/ Server	3x4 email + 4x3 pass + + 1x1 pin = 25	3x1 email + 1x 1 pin = 4
Multi-cloud w/ Password manager	3x4 email + 4x4 pass = 28	3x1 email + 4x1 pass = 7
Multi-cloud w/ Secure enclave	3x4 email + 4x4 pass = 28	3x1 email + 4x1 pass = 7
Centralised Server	3x1 email + 4x1 pass = 7	0
Web Local Storage	4x1 pass = 4	4x1 pass = 4
Hardware Wallet	2x1 pin = 2	2x1 pin = 2

Figure 27: Credential management effort for each approach (lower is better)

Approaches	Without password manager	With password manager
Multi-cloud w/ Server	9 clicks + 25 = 34	6 clicks + 4 = 10
Multi-cloud w/ Password manager	9 clicks + 28 = 37	6 clicks + 7 = 13
Multi-cloud w/ Secure enclave	9 clicks + 28 = 37	6 clicks + 7 = 13
Web Local Storage	1 click + 4 = 5	1 click + 4 = 5
Hardware Wallet	15 clicks + 2 = 17	15 clicks + 2 = 17

Figure 28: Combined user effort when signing a transaction for each approach (lower is better)

As a conclusion in terms of usability, the web local storage approach is superior to the others, but the multi-cloud with server only lacks behind on the number of clicks required to sign a transaction, as the final score is close between them. The other multi-cloud approaches are consistently in third place and the hardware wallet in last.

## 6.2 SECURITY

The attack vectors that will be considered in this comparison are:

- The device that the user uses to access the wallet becoming inaccessible (because of loss, theft, failure, etc.);
- The server that the wallet uses (or not) to store information is compromised and accessed by external people;
- The private key being exposed in memory while a transaction is signed, which may be possible to explore as an attack vector;
- Having physical access to the user's device with the wallet unlocked.

Approaches	Inaccessible locked device (loss/theft/failure)	Compromised server	Compromised device	
			PK exposed in memory client side	Access unlocked account
Multi-cloud w/ Server	✓	🔒	yes	✓
Multi-cloud w/ Password manager	✓	🔒	yes	✓
Multi-cloud w/ Secure enclave	🔒	✓	yes	✓
Web Local Storage	🔒	✓	yes	⚠️
Hardware Wallet	🔒	✓	✓	⚠️

🔒 - possible to backup with seed phrase  
 ⚠️ - possible total loss of the wallet  
 ✓ - not vulnerable

Figure 29: Possible attack vectors and approach vulnerability, along with potential damage and whether mitigation is possible

Secure enclave, hardware wallet, and local web storage approaches are all vulnerable to loss, theft, or a hardware failure that may be critical to access the wallet, but it does not imply losing the funds completely, because the user may still have the backup seed phrase that can restore the wallet. As for a compromised server, the multi-cloud approaches that use one are not vulnerable, as the server does not store the private key and the user could use the backup seed phrase to restore the wallet. Only in the hardware wallet, the private key is never exposed in memory during the signing of the transaction, as the signing is not performed on the client-side. This is a possible vulnerability of the other approaches.

Access to an unlocked device with the wallet unlocked, may lead to total loss in the non multi-cloud approaches. In these, the user is required to introduce a password or pin code when signing a transaction. In the others, after the wallet is unlocked, the user is able to sign transactions without having to unlock it again. Figure 29 summarizes the results of systematic security analysis, showing the possible impact of each attack on each approach.

With the goal of making an objective security evaluation, multiple techniques and tools that could possibly compromise the proposed approaches were studied. Some require access to the machine unlocked and other could be performed remotely with malicious software inadvertently installed by the user.

### 6.2.1 Memory Exfiltration

Two tools were used to evaluate memory exfiltration: Chrome developer tools and Mimikatz. Both of these require full control and admin permissions of the device, remote or physical. The Chrome developer tools (*chr*), are used to evaluate if the private key of a user can be found at any point of the signing of a transaction on the client-side. We took a browser's memory snapshot during the signing of the transaction. We were not able to find the exposed private key in any object of the snapshot. We have done the same experiment without erasing the private key from memory (in the multi-cloud approaches) and we noticed it was exposed, so, we believe the method of changing its value to null is effective when protecting the key from an exploit with some machine access. To use this as an attack would require remote or physical access.
















Mimikatz is a widely used tool that can be used to explore the memory of the operating system and applications. It only works on Windows operating systems and it can be used to steal authentication credentials, passwords, and others. In this context, it was used to see if it was possible to explore Google Chrome extensions data, in order to find the private key of a Metamask account. This was unsuccessful, as we believe it is only possible to explore passwords stored by the Chrome password manager, as seen in Figure 30. This is important because, in the previous usability discussion, we have mentioned the usage of a password manager. This specific password manager may be vulnerable if attacked with this tool. As seen in (Chen et al., 2012), browser embedded password managers are possibly vulnerable to self-exfiltration attacks.

### 6.2.2 Stealing credentials

In a situation where a user would leave his device unlocked momentarily and someone could access it with the unlocked wallets, the web local storage approach would be vulnerable because after initially unlocking the account, the wallet does not require the password for any other action. The attacker could export the private key or sign transactions. As for the others, they require always extra authentication at the moment of the signature, such as a password or a pin code. So, even if the cloud accounts were to be unlocked, the attacker still could not sign a transaction or access the funds with the multi-cloud or the hardware wallet approaches, as in Figure 30.

Another important aspect when evaluating security, is the number of different platforms and passwords that must be compromised in order to perform a harmful attack and if access to the device remotely or physical is necessary. Figure 31 presents the number of credentials required and if these were to be obtained, which degree of access to the machine is necessary to perform an attack.

In the case of the web local storage approach, a single password is required to be compromised to have full access to the wallet with access to the device, both physical or remote. It would also be possible to access the wallet without access to the machine if the cache file where the encrypted data is stored could be obtained. This file is unprotected and located in a browser's folder. With both of these, it is possible to use the vault decryptor tool (*met*), which exposes the mnemonic phrase that can be used to access the wallet on any device. In multi-cloud approaches, it is necessary to compromise three different cloud providers and an extra platform, which clearly shows the advantage of a multi-signature approach. The server and the password manager require

Approaches	Without device access	With device access (physical/remote)	
	Key logger w/ file permissions	Memory Exfiltration	Access unlocked account
Multi-cloud w/ Server			
Multi-cloud w/ Password manager			
Multi-cloud w/ Secure enclave			
Web Local Storage			
Hardware Wallet			



 - Vulnerable       - Not vulnerable

Figure 30: Approach vulnerability to different techniques

authentication credentials but do not require any further access to the machine itself. The secure enclave would require physical or remote access to the device. When something is to be read from the secure enclave, the user needs to make an authentication, so it would be necessary to obtain the credentials and insert them to access the keys. To compromise the hardware wallet approach it would be necessary to have physical access to the device and access to the pin code.

A known common attack used to steal credentials is the key logger, which records the keystrokes of the keyboard of a user and sends them to the attacker. This is possible if the user installed a malicious program on its machine, for example. It does not need device access, because the user may install this program by himself. Approaches that require physical hardware or the web local storage approach, that uses a local file, are not vulnerable to this attack in its basic form. But, it could be assumed that if the program has the ability to record the keystrokes would also have permission to access unprotected files on the machine. If that was the case, then the web local storage approach would be compromised, as was explained above. The multi-cloud with server and password manager approaches are vulnerable, as the accounts needed to be accessed are all protected with credentials. If these are stolen, the attacker can access the wallet from any device, as seen in Figure 30.

Approaches	Number of passwords/platforms needed to explore	Full physical access with credentials	Full remote access with credentials	Only credentials with no access
Multi-cloud w/ Server	4 credentials (email/password or pin)	!	!	!
Multi-cloud w/ Password manager	4 credentials (email/password)	!	!	!
Multi-cloud w/ Secure enclave	4 credentials (email/password)	!	!	✓
Web Local Storage	1 credential (password) 1 cache file	!	!	✓
Hardware Wallet	1 credential (pin)	!	✓	✓



 - Vulnerable     
  - Not vulnerable

Figure 31: Required credential compromise and access for successfully exploiting each approach

### 6.2.3 Overall conclusions on wallet security

The final conclusion based on Figure 30 and Figure 31 show that the hardware wallet performed the best, which was an expected result. The next best results are from the multi-cloud secure enclave approach, which is not vulnerable to any of the selected tools and would require access to the device plus the credentials to perform a successful attack. Between the web local storage and the other multi-cloud approaches, the web local storage approach would be vulnerable to physical access to the unlocked account and fewer credentials would need to be compromised when compared with the multi-cloud. So, we can conclude that the multi-signature approaches offer stronger security because the attacker needs to succeed in exploiting multiple platforms.

These results help us conclude that having a separate device or secure enclave storing a cryptographic key increases the security substantially. While having everything stored on the cloud or encrypted data on a file may be dangerous if the user has malicious software installed with the purpose of stealing a specific type of wallet. But, in this case, the increased number of credentials implies having better security.

## 6.3 COMBINED USABILITY AND SECURITY ASSESSMENT

As a final summary, multi-cloud approaches are the most balanced in security and usability, performing well in both. The multi-cloud with server approach has an advantage in usability, but performs identically to the multi-cloud with password manager approach in terms of security, and is outperformed by the secure enclave approach. This is mainly due to it being reliant on physical hardware similar to the hardware wallet, which is



the best in terms of security and the last in usability. The web local storage approach performs similarly to the multi-cloud with server in terms of usability but is inferior in security.

To conclude, we consider the multi-cloud approaches the most balanced and believe they could be an important leap in the market of non-technical users, that is still not much explored. Being able to provide this amount of security with little usability compromises could help prevent many wallet extorsions and attract many users to this growing market.

---

## CONCLUSIONS AND FUTURE WORK

---

In this dissertation, it was intended to combine different positive aspects of multiple crypto wallet implementations in a single product and provide its basic implementation. This would be objectively compared in security and usability to other products in the market.

We started by studying the various existent crypto wallet approaches, as well as finding the ones that have the bigger share of the market. We have concluded that the web wallets are the most used and that blockchain interaction is an important characteristic of a wallet. It was also observed that multi-cloud was a little-explored technique that could bring something new to the market. From that study, we conceived three multi-cloud approaches that were implemented and built on top of Metamask. These were essential to achieve our proposed goal and a big contribution of our work. The frameworks used to evaluate security and usability of the crypto wallets were built and proposed by us.

After performing objective tests using existing tools and known studies, we concluded that the multi-cloud approaches were the most balanced in security and usability. They were not the best performers in security or usability, as these places were occupied by hardware wallets and web local storage respectively. However, the multi-cloud approaches performed very positively in security with very few compromises in usability, almost matching the web local storage.

In a continuation of this work, it would be possible to propose more approaches and test them against the same parameters and tests. An interesting proposal would be the usage of the personal smartphone, which could possibly safeguard memory exploitation attacks and have a better security score. As most people already carry a smartphone, this would not be considered an extra device needed and would not have as big of an impact as the hardware wallet has on usability. It would be similar to the multi-cloud with secure enclave approach but implemented on mobile. Although this approach was not implemented or explored in this work, due to time constraints, we think it is worth exploring in future work.

To improve the proposed implementations and turn them into finished products, it is necessary to implement the multi-cloud connection website with more cloud providers. In our simplified version, it was only used one cloud.

In conclusion, we have achieved the goals proposed in the beginning, providing three basic implementations of the new cryptographic key management approach, in the form of crypto wallets. We have also provided an extensive objective comparison in a contemporary and ongoing subject that is still very unexplored and believed to have an ambitious future.

---

## BIBLIOGRAPHY

---

- Chrome devtools. URL <https://developer.chrome.com/docs/devtools/>.
- Metamask vault decryptor. URL <https://metamask.github.io/vault-decryptor/>.
- M. A. AlZain, E. Pardede, B. Soh, and J. A. Thom. Cloud computing security: From single to multi-clouds. pages 5490–5499, 2012. doi: 10.1109/HICSS.2012.153.
- Fiolitakis Antonios, Nikolaos Petrakis, Panagiotis Margaronis, and Emmanuel Antonidakis. Hardware implementation of triple-des encryption/decryption algorithm. 01 2006.
- Apple. Keychain data protection overview. 12 2020. URL <https://support.apple.com/lt-1t/guide/security/secb0694df1a/1/web/1>.
- J. Attridge. An overview of hardware security modules. 2019.
- F. J. Aufa, Endroyono, and A. Affandi. Security system analysis in combination method: Rsa encryption and digital signature algorithm. In *2018 4th International Conference on Science and Technology (ICST)*, pages 1–5, 2018. doi: 10.1109/ICSTC.2018.8528584.
- AWS. Aws cloud hsm. 12 2020a. URL <https://aws.amazon.com/cloudhsm/>.
- AWS. Aws key management services. 12 2020b. URL <https://aws.amazon.com/kms/>.
- Ken Beer and Ryan Holland. Encrypting data at rest. *White Paper of amazon web services*, 2014.
- Steven M. Bellovin. Frank miller: Inventor of the one-time pad. *Cryptologia*, 35(3):203–222, 2011. doi: 10.1080/01611194.2011.583711. URL <https://doi.org/10.1080/01611194.2011.583711>.
- Rajdeep Bhanot and Rahul Hans. A review and comparative analysis of various encryption algorithms. *International Journal of Security and Its Applications*, 9:289–306, 04 2015. doi: 10.14257/ijjsia.2015.9.4.27.
- B. Bhat, A. W. Ali, and A. Gupta. Des and aes performance evaluation. pages 887–890, 2015. doi: 10.1109/CCAA.2015.7148500.
- Binance. What is a multisig wallet? *Binance Academy*, 12 2020a. URL <https://academy.binance.com/en/articles/what-is-a-multisig-wallet>.
- Binance. Crypto wallet types explained. *Binance Academy*, 12 2020b. URL <https://academy.binance.com/en/articles/crypto-wallet-types-explained>.

- Blockchain.com. Blockchain.com wallets. 01 2021. URL <https://www.blockchain.com/charts/my-wallet-n-users>.
- Y. E. Bulut. Secure hardware cryptocurrency wallet within common criteria framework. 2019.
- S. Chandra, S. Paira, S. S. Alam, and G. Sanyal. A comparative survey of symmetric and asymmetric key cryptography. pages 83–93, 2014a. doi: 10.1109/ICECCE.2014.7086640.
- S. Chandra, S. Paira, S. S. Alam, and G. Sanyal. A comparative survey of symmetric and asymmetric key cryptography. pages 83–93, 2014b. doi: 10.1109/ICECCE.2014.7086640.
- Eric Y Chen, Sergey Gorbaty, Astha Singhal, and Collin Jackson. Self-exfiltration: The dangers of browser-enforced information flow control. In *Proceedings of the Workshop of Web*, volume 2. Citeseer, 2012.
- Jaeik Cho, Setiawan Soekamtoputra, Ken Choi, and Jongsub Moon. Power dissipation and area comparison of 512-bit and 1024-bit key aes. *Computers Mathematics with Applications*, 65(9):1378 – 1383, 2013. ISSN 0898-1221. doi: <https://doi.org/10.1016/j.camwa.2012.01.035>. URL <http://www.sciencedirect.com/science/article/pii/S0898122112000454>. Advanced Information Security.
- Nakhoon Choi and Heeyoul Kim. A blockchain-based user authentication model using metamask. *Journal of Internet Computing and Services*, 20(6):119–127, 2019.
- Coinbase. About coinbase. 01 2021. URL <https://www.coinbase.com/about>.
- D. Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM Journal of Research and Development*, 38(3):243–250, 1994. doi: 10.1147/rd.383.0243.
- Túlio Cicero Salvaro de Souza, Jean Everson Martina, and Ricardo Felipe Custódio. Audit and backup procedures for hardware security modules. page 89–97, 2008. doi: 10.1145/1373290.1373302. URL <https://doi.org/10.1145/1373290.1373302>.
- Vincenzo Di, Riccardo Longo, Federico Mazzone, and Gaetano Russo. Resilient custody of crypto-assets, and threshold multisignatures. *Mathematics*, 8:10, 10 2020. doi: 10.3390/math8101773.
- Y. Dodis and J. Spencer. On the (non)universality of the one-time pad. pages 376–385, 2002. doi: 10.1109/SFCS.2002.1181962.
- R. Dowsley, M. Gabel, G. Hubsch, G. Schiefer, and A. Schwichtenberg. A distributed key management approach. pages 509–514, 2016. doi: 10.1109/CloudCom.2016.0089.
- W. Fumy and P. Landrock. Principles of key management. *IEEE Journal on Selected Areas in Communications*, 11(5):785–793, 1993. doi: 10.1109/49.223881.
- Kris Gaj and Pawel Chodowiec. Comparison of the hardware performance of the aes candidates using reconfigurable hardware. In *AES Candidate Conference*, pages 40–54, 2000.

- Google. Google cloud hsm. 12 2020a. URL <https://cloud.google.com/kms/docs/hsm>.
- Google. Google cloud key management services. 12 2020b. URL <https://cloud.google.com/security-key-management>.
- K. Gupta and S. Silakari. Performance analysis for image encryption using ecc. pages 79–82, 2010. doi: 10.1109/CICN.2010.26.
- Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. (non-) random sequences from (non-) random permutations—analysis of rc4 stream cipher. *Journal of Cryptology*, 27(1):67–108, 2014.
- M. Guri. Bitcoin: Leaking private keys from air-gapped cryptocurrency wallets. pages 1308–1316, 2018. doi: 10.1109/Cybermatics\_2018.2018.00227.
- Juhyeng Han, Seongmin Kim, Taesoo Kim, and Dongsu Han. Toward scaling hardware security module for emerging cloud services. In *Proceedings of the 4th Workshop on System Software for Trusted Execution, SysTEX '19*, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368889. doi: 10.1145/3342559.3365335. URL <https://doi.org/10.1145/3342559.3365335>.
- A. A. Hasib and A. A. M. M. Haque. A comparative study of the performance and security issues of aes and rsa cryptography. 2:505–510, 2008. doi: 10.1109/ICCIT.2008.179.
- D. He, S. Li, C. Li, S. Zhu, S. Chan, W. Min, and N. Guizani. Security analysis of cryptocurrency wallets in android-based applications. *IEEE Network*, 34(6):114–119, 2020. doi: 10.1109/MNET.011.2000025.
- Kipp E.B. Hickman. The ssl protocol. 1994. URL <http://www.webstart.com/jed/papers/HRM/references/ssl.html>.
- Horizen. Types of wallets. *Horizen Academy*, 12 2020. URL <https://academy.horizen.io/technology/advanced/types-of-wallets/>.
- X. Huang and R. Chen. A survey of key management service in cloud. pages 916–919, 2018. doi: 10.1109/ICSESS.2018.8663805.
- ICOHolder. The most comprehensive metamask wallet review. 12 2020. URL <https://icoholder.com/blog/metamask-wallet-review/>.
- T. P. Innokentievich and M. V. Vasilevich. The evaluation of the cryptographic strength of asymmetric encryption algorithms. pages 180–183, 2017. doi: 10.1109/RPC.2017.8168094.
- Stevo Jokic. Analysis and security of crypto currency wallets. *ZBORNIK RADOVA UNIVERZITETA SINERGIJA*, 19, 05 2019. doi: 10.7251/ZRSNG1801102J.
- S. Kansal and M. Mittal. Performance evaluation of various symmetric encryption algorithms. pages 105–109, 2014. doi: 10.1109/PDGC.2014.7030724.

- Ambarish Karole, Nitesh Saxena, and Nicolas Christin. A comparative usability evaluation of traditional password managers. In *International Conference on Information Security and Cryptology*, pages 233–251. Springer, 2010.
- Keyruptive. Keyruptive. 01 2021. URL <https://keyruptive.com>.
- Benjamin K Kikwai. Elliptic curve digital signatures and their application in the bitcoin crypto-currency transactions. *International Journal of Scientific and Research Publications*, 7(11):135–138, 2017.
- P. Kitsos, G. Kostopoulos, N. Sklavos, and O. Koufopavlou. Hardware implementation of the rc4 stream cipher. 3:1363–1366 Vol. 3, 2003. doi: 10.1109/MWSCAS.2003.1562548.
- Talia Knowles-Rivas. Metamask exceeds 1 million monthly active users. 01 2021. URL <https://medium.com/metamask/metamask-exceeds-1-million-monthly-active-users-9da72a1e915d>.
- B. Köppel and S. Neuhaus. Analysis of a hardware security module’s high-availability setting. *IEEE Security Privacy*, 11(3):77–80, 2013. doi: 10.1109/MSP.2013.56.
- Ledger. The company. 01 2021. URL <https://www.ledger.com/the-company>.
- Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor’s new password manager: Security analysis of web-based password managers. pages 465–479, August 2014. URL [https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/li\\_zhiwei](https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/li_zhiwei).
- Dindayal Mahto and DILIP YADAV. Rsa and ecc: A comparative analysis. *International Journal of Applied Engineering Research*, 12:9053–9061, 01 2017.
- Itsik Mantin and Adi Shamir. *A Practical Attack on Broadcast RC4*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.
- C. Matt and U. Maurer. The one-time pad revisited. pages 2706–2710, 2013. doi: 10.1109/ISIT.2013.6620718.
- MetaMask. Metamask/metamask-extension. URL <https://github.com/MetaMask/metamask-extension>.
- Metamask. Metamask. 12 2020. URL <https://metamask.io/index.html>.
- Evgeny Milanov. The rsa algorithm. 2009. URL [https://sites.math.washington.edu/~morrow/336\\_09/papers/Yevgeny.pdf](https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf).
- Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009. URL <http://www.bitcoin.org/bitcoin.pdf>.

- T. Nie and T. Zhang. A study of des and blowfish encryption algorithm. pages 1–4, 2009. doi: 10.1109/TENCON.2009.5396115.
- Om Pal, Bashir Alam, Vinay Thakur, and Surendra Singh. Key management for blockchain technology. *ICT Express*, 2019. ISSN 2405-9595. doi: <https://doi.org/10.1016/j.icte.2019.08.002>. URL <http://www.sciencedirect.com/science/article/pii/S2405959519301894>.
- M. Panda. Performance analysis of encryption algorithms for security. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPE5)*, pages 278–284, 2016. doi: 10.1109/SCOPE5.2016.7955835.
- R. Pich, S. Chivapreecha, and J. Prabnasak. A single, triple chaotic cryptography using chaos in digital filter and its own comparison to des and triple des. pages 1–4, 2018. doi: 10.1109/IWAIT.2018.8369682.
- Rogério Pontes, Dorian Burihabwa, Francisco Maia, João Paulo, Valerio Schiavoni, Pascal Felber, Hugues Mercier, and Rui Oliveira. Safefs: A modular architecture for secure user-space file systems: One fuse to rule them all. In *Proceedings of the 10th ACM International Systems and Storage Conference, SYSTOR '17*, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350358. doi: 10.1145/3078468.3078480. URL <https://doi.org/10.1145/3078468.3078480>.
- Sabrina J. Poon, Margaret B. Greenwood-Ericksen, Rebecca E. Gish, Pamela M. Neri, Sukhjit S. Takhar, Scott G. Weiner, Jeremiah D. Schuur, and Adam B. Landman. Usability of the massachusetts prescription drug monitoring program in the emergency department: A mixed-methods study. *Academic Emergency Medicine*, 23(4):406–414, 2016. doi: <https://doi.org/10.1111/acem.12905>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/acem.12905>.
- N. R. Potlapally, S. Ravi, A. Raghunathan, and G. Lakshminarayana. Optimizing public-key encryption for wireless clients. In *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333)*, volume 2, pages 1050–1056 vol.2, 2002. doi: 10.1109/ICC.2002.997014.
- T. Prasad, P.s.R. Chandra, Srinivasa Manyala, and g rao. Analysis of brute-force attack in ues over des. 8: 423–425, 04 2019.
- J. Raigoza and K. Jituri. Evaluating performance of symmetric encryption algorithms. pages 1378–1379, 2016. doi: 10.1109/CSCI.2016.0258.
- Dirk Rijmenants. Is one-time pad history. *Cipher Machines & Cryptology*, pages 1–4, 2009.
- Dr-Diaa Salama, Hatem Abd elkader, and Mohiy M. Hadhoud. Performance evaluation of symmetric encryption algorithms. *Communications of the IBIMA*, 10, 01 2009.
- Gustavus J. Simmons. Symmetric and asymmetric encryption. *ACM Comput. Surv.*, 11(4):305–330, December 1979. ISSN 0360-0300. doi: 10.1145/356789.356793. URL <https://doi.org/10.1145/356789.356793>.

- Gurpreet Singh and Supriya Kinger. A study of encryption algorithms (rsa, des, 3des and aes) for information security. *International Journal of Computer Applications*, 67:33–38, 04 2013. doi: 10.5120/11507-7224.
- P. Singh and K. Kaur. Database security using encryption. pages 353–358, 2015. doi: 10.1109/ABLAZE.2015.7155019.
- S. R. Singh, A. K. Khan, and S. R. Singh. Performance evaluation of rsa and elliptic curve cryptography. pages 302–306, 2016. doi: 10.1109/IC3I.2016.7917979.
- O. Srinivasa. Performance analysis of des and triple des. *International Journal of Computer Applications*, 130: 30–34, 11 2015. doi: 10.5120/ijca2015907190.
- Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. Lattice-based threshold changeability for standard shamir secret-sharing schemes. *IEEE Transactions on Information Theory*, 53(7):2542–2559, 2007. doi: 10.1109/TIT.2007.899541.
- J. Thakur and Nagesh Kumar. Des, aes and blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International Journal of Emerging Technology and Advanced Engineering*, 1:6–12, 01 2011.
- Kanwardeep Singh Walia, Shweta Shenoy, and Yuan Cheng. An empirical analysis on the usability and security of passwords. In *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 1–8, 2020. doi: 10.1109/IRI49571.2020.00009.
- M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh. Comprehensive study of symmetric key and asymmetric key encryption algorithms. pages 1–7, 2017. doi: 10.1109/ICEngTechnol.2017.8308215.
- K. Zeng, C. . Yang, D. . Wei, and T. R. N. Rao. Pseudorandom bit generators in stream-cipher cryptography. *Computer*, 24(2):8–17, 1991. doi: 10.1109/2.67207.
- F. Zhu, W. Chen, Y. Wang, P. Lin, T. Li, X. Cao, and L. Yuan. Trust your wallet: A new online wallet architecture for bitcoin. pages 307–311, 2017. doi: 10.1109/PIC.2017.8359562.