Miguel Afonso Madeira

Licenciado em Engenharia Informática

Towards more Secure and Efficient Password Databases

Relatório intermédio para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Bernardo Ferreira, Invited Assistant Professor, NOVA University of Lisbon

Co-orientador: João Leitão, Assistant Professor, NOVA University of Lisbon

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Fevereiro, 2018

Abstract

Password databases form one of the backbones of nowadays web applications. Every web application needs to store its users' credentials (email and password) in an efficient way, and in popular applications (Google, Facebook, Twitter, etc.) these databases can grow to store millions of user credentials simultaneously. However, despite their critical nature and susceptibility to targeted attacks, the techniques used for securing password databases are still very rudimentary, opening the way to devastating attacks. Just in the year of 2016, and as far as publicly disclosed, there were more than 500 million passwords stolen in internet hacking attacks.

To solve this problem we commit to study several schemes like property-preserving encryption schemes (e.g. deterministic encryption), encrypted data-structures that support operations (e.g. searchable encryption), threshold cryptography (e.g. secret sharing), fully/partially homomorphic encryption schemes, and commodity trusted hardware (e.g. Intel SGX).

In this thesis we propose to make a summary of the most efficient and secure techniques for password database management systems that exist today and recreating them to accommodate a new and simple universal API. Additionally, in this thesis we will also build two new techniques for secure password database management, one based on cryptographic encryption and the other based on commodity trusted hardware. These solution will preserve security guarantees while allowing users to login under the "one second" usability mark.

Furthermore, since password databases are particularly susceptible to snapshot adversaries (i.e. a hacker that gets a small time window of access to the database and makes a snapshot copy of all available information), the thesis will also study this adversary in detail and research how to efficiently counter it.

Keywords: Web Applications, Password Databases, Snapshot Attacker, Searchable Encryption, Trusted Hardware. ...

Resumo

Base de dados de passwords formam o esqueleto de aplicações web atuais. Todas as aplicações web necessitam de armazenar as credenciais do utilizador (email e password) de uma maneira eficiente, e em aplicações populares (Google, Facebook, Twitter, etc.) estas bases de dados podem crescer e ter que armazenar milhões de credenciais simultaneamente. No entanto, apesar da sua natureza crítica e susceptibilidade a ataques, as técnicas usadas para armazenar passwords continuam a ser muito rudimentares, abrindo o caminho para ataques devastadores. Só no ano de 2016, e apenas os números partilhados, houve mais de 500 milhões de passwords roubadas devido a ataques de hackers.

Para resolver este problema estamos determinados en estudar os principais esquemas de cifra existentes, tais como esquemas criptográficos que preservam propriedades (por ex. Cifra Determinista), estruturas de dados cifradas que suportam operações (por ex. Cifra Pesquisável), Criptografia de Limite (por ex. Partilha de Segredos), Encriptação Homomórfica Completa e Parcial, e Hardware Confiável (por ex. Intel SGX).

Nesta tese propomos fazer um resumo das técnicas mais eficientes e seguras para base de dados de passwords que existem hoje em dia, e recriá-las para acomodar uma nova e simples API universal. Adicionalmente, nesta tese iremos também construir duas novas técnicas para base de dados de password, uma baseada em esquemas criptográficos e outra em hardware confiável, preservando segurança e permitindo que o utilizador faça operações debaixo da marca de "um segundo"de usabilidade.

Além disso, como as base de dados de password continuam a ser susceptíveis a snapshot attacks (por ex. um adversário que consegue acesso durante uma pequena janela de tempo e faz uma copia de toda a informação disponível), esta tese também irá estudar o adversário em detalhe e fazer um estudo sobre como defender de ataques eficientemente.

Palavras-chave: Aplicações Web, Base de Dados de Passwords, Ataques Snapshot, Cifra Pesquisável, Hardware Confiável ...

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

As the access to the Internet continues to grow everyday, more and more people around the globe tend to join it. More people means increased storage capacity needed to store more of the user's personal data in popular applications like Google, Twitter, Facebook, Instagram, just to name a few. Each of these websites contains sensible information about their users, which is usually protected by a username and a password. That's what always separated ones personal information from the rest of the world, their password, just like your house key protects your house from everybody coming in uninvited, the users password supposedly is required and mandatory for them to access their account in web applications.

1.1 Problem Statement

Despite their critical nature and susceptibility to targeted attacks, the techniques used for securing password databases are still very rudimentary, opening the way to devastating attacks. Adversaries, like hackers, tend to always explore new and innovative ways to attack and access the data, and unfortunately they're still very successful.

Just in the year of 2016, and as far as publicly disclosed, there were more than 500 million passwords stolen in Internet hacking attacks [47]. Last year on 2017 all 3 billion of Yahoo accounts were breached [28]. This accounted for the users Full Name, Usernames and Password.

Password databases are particularly susceptible to snapshot attackers, which is a hacker that gets a small time window of access to the database and makes a

snapshot copy of all available information. If the data in the database storage is not encrypted, and is just in plain text, the adversary can just look at it and learn all the user's credential data. If the users credentials in the snapshot copy are hashed, the adversary can just run a Brute Force or a Dictionary Attack, among others to discover the original plaintext that was hashed.

One used solution to this problem is to encrypt the users credential data, and every time the user wishes to enter the system, she provides its username and password. The system then decrypts its stored data with the corresponding key and compares the decrypted plaintext user's password with the user's entered password. One problem with this solution is that there's a time frame in which the original password is in its plaintext format, for the system to make the comparison between the password values, making the system vulnerable to an attack in this time frame, as the adversary can create a snapshot at this moment to gain the original plaintext password.

One other used solution is to encrypt the data in a way that allows computations over the encrypted. However this method is very expensive and somewhat slow, causing it to be inefficient for the users to make operations like logging in or registering.

These are the reasons some websites still store the users passwords with just the hashed values of them, or just in plaintext, to remain fast and simple for the system when a user wants to complete a login operation.

So in this thesis we will try to answer the following question:

Can we build a secure login system that once attacked by an adversary can't be compromised and still be efficient enough to stay under the "one second" usability mark?

1.2 Objectives and Expected Contributions

This thesis main objective is to build more secure and efficient methods of password database management. For this reason the thesis will focus on the study of previous password database methods, in order to grow a deeper understanding of the existing scientific solutions, regarding a secure and efficient way to store and access the users passwords. By studying the current situation, and already proofed secure methods, we will have more knowledge on the subject and accomplish a bigger contribution to the scientific security world.

In the list below we enumerate the planned work in the construction of this thesis:

- Build a tool that allows to test several operations (like login, register a user, etc..) in password database methods that already exist today. The tests will be based on time and space complexities, with a variable number of entries.

- The Design and implementation of new mechanisms to protect password databases, based on cryptography.

- Design and implementation of a password database management of a system, that would not be based on cryptography, but based on trusted hardware, like INTEL SGX.

- Extensively evaluate the performance of the previous contributions and compare them to already existing methods, using the tool mentioned above.

- Evaluate the security of the developed contributions and analyse their resistance to adversarial attacks.

In the end of the work in this thesis, it is expected that we have at least 2 different solutions to be presented, that are proofed to be more secure and efficient then current and past methods used in the industry. For testing efficiency we will measure the time and space complexity. For time complexity we will measure on how much time it takes for the system, since a user - password is inputted, to compare it to the ciphertext and to output the result of the comparing operation. To measure space complexity we calculate how much space a set of passwords will take on the systems database. These results will of these operation will then be compared to the existing and most common used methods today.

1.3 Report Organization

The remaining of this report is organized in the following structure:

- **Chapter 2:** In this chapter we present the important background and related work that includes fundamental concepts necessary to the understanding of the concept of this thesis.

- **Chapter 3:** In this chapter we present the methods involved in the making of the final product of the thesis, like the initial view of the system model and architecture. Here we will also state more specifically what are we aiming to accomplish on this work.

- **Chapter 4:** In this chapter we present the work plan for the thesis, describing the planned activities and how much estimated time which one will take, while presenting a Gantt chart.

CHAPTER 2

Background and related work

In this chapter, we discuss the relevant background and related work with importance for this thesis. The chapter is divided into three sections. In Section 2.1 we present several ways to make operations on encrypted data on a cloud server. In Section 2.2 we present Searchable Encryption and Searchable Symmetric Encryption. In section 2.3 we present the state of the art in password databases and also present the most common attacks made by adversaries on those.

2.1 Computation On Encrypted Data

Everyday cloud computing gets faster, more available and cheaper, so it's only natural that extra users are joining this types of services to store their data. Cloud computing enable network on demand and convenient access to a shared pool of configurable computing resources like networks, servers, storage, applications and services [35] and can be provisioned and released with minimal user effort or interaction with the service provider. But the main advantage of cloud computing is that is accessible from almost anywhere in the world, at any time. As long as there is Internet access, the user can access the services of the cloud to outsource his data and later can request to access it and make computations over it. Cloud users include private parties and also companies, that use this system as the primary focus to their business models, so they don't have to expand their resources to servers infrastructures and have the extra cost of maintaining them.

The problem with cloud computing is that an unwanted party, like the manufacturer, a software engineer or even an Internet attacker, can see the users data if it

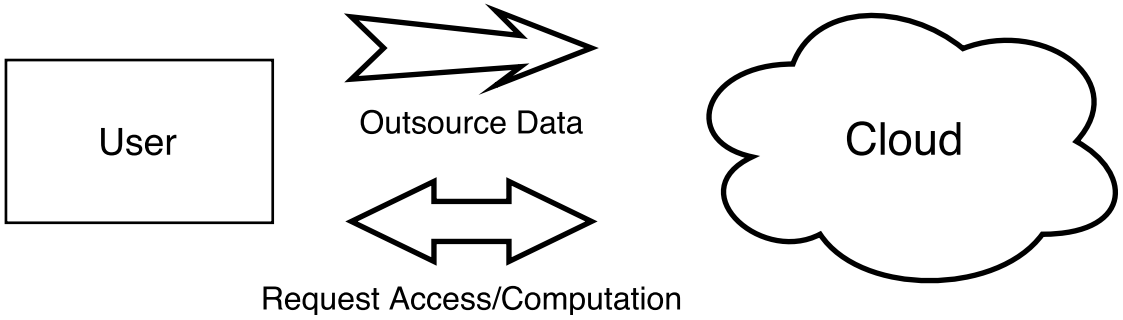


Figure 2.1: User-Cloud Interaction

is not encrypted. That's why the data stored in the storage cloud should always be encrypted, but this raises the problem that computations can't be simply done in an encrypted form.
Computation over encrypted data introduces several cryptographic methods to securely perform computations without revealing private and sensitive information to the cloud, and without requiring the user's secret key. This means that the user or the cloud server doesn't have to decrypt the whole encrypted data to perform a desired function [21], as the computations are done on the encrypted form of the data.
In the following sub-sections we are going to discuss Property-Preserving Encryption (Deterministic Encryption and Order Preserving Encryption), Homomorphic Encryption (Fully Homomorphic Encryption and Partial Homomorphic Encryption) and Computations on Trusted Hardware (TPM, Intel SGX and ARM TrustZone).

2.1.1 Property-preserving Encryption

Property-preserving Encryption enables some properties to be preserved on encrypted data. For this the algorithm has to preserve some properties of the encrypted data, such as its order. These associated properties allow for some pre-determined computation on the encrypted data such as range queries, even if the data was generated independently and individually [40]. In the following sections we present two types of Property-Preserving Encryption.

2.1.1.1 Deterministic Encryption

Deterministic Encryption is a cryptosystem, that given a certain plaintext and key, always produces the same exact ciphertext, even over separate executions.

We can see an example of Deterministic Encryption in **Figure 2.2**. In this example there are two different runs of Deterministic Encryption, in the first run we input

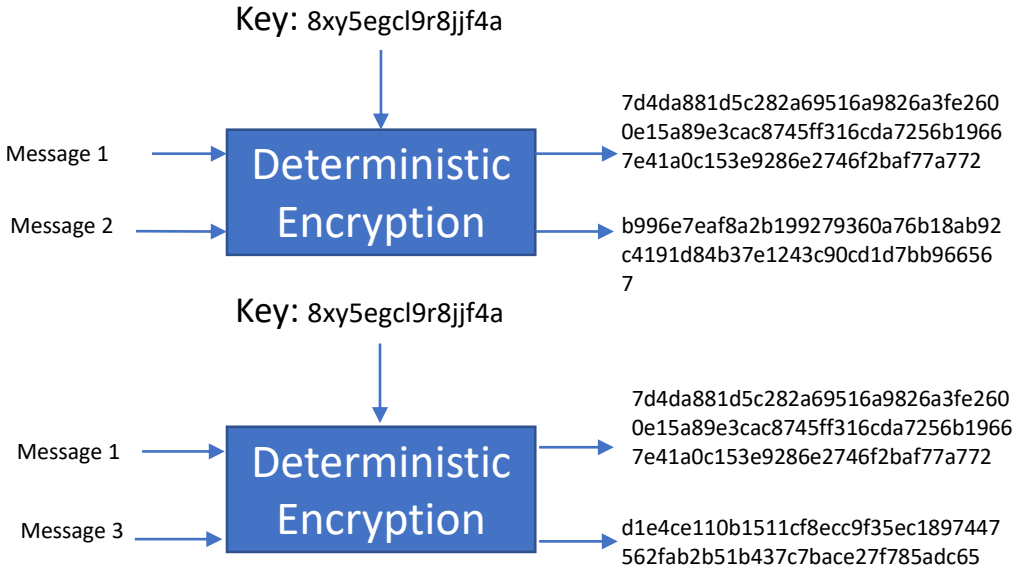


Figure 2.2: Example of Deterministic Encryption

Message1 and Message2 along with a key, and get a different result for each one. In the second run we input the same Message1, and Message3, with the same key as the first run and also get a pair of results. As we can note, the output for the same input (Message1 + key) is the same for each run and when the input is different, the output will also be different (see the output of Message2 and Message3).

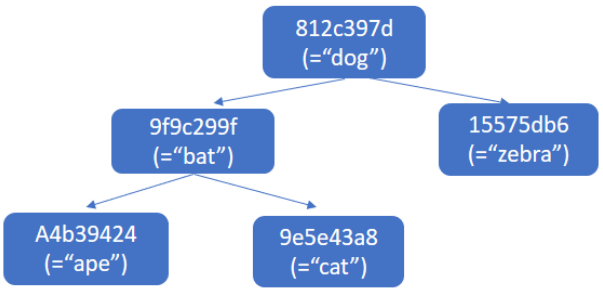
2.1.1.2 Order-Preserving Encryption

Order preserving encryption is a deterministic encryption scheme, that preservers the order of the numerical plaintext [5]. This is useful because it allows efficient range queries on encrypted data. This way an untrusted database server can index sensitive data in encrypted form, inside a data structure (such as a b-tree) permitting range queries (for example, from a to b), to locate the desired function in logarithm time. Basically, given two plaintexts x and y the definition of OPE can be given as follows:

$$x > y -> encryption(x) > encryption(y).$$

In figure **Figure 2.3** we can see a example of how to store the encrypted data with OPE in a database storage. The figure contains a reference table and a b-tree. The reference table contains the hexadecimal value of every plaintext and also the ciphertext, that is the result of deterministic encryption on the plaintext, so the algorithm can calculate where on the b-tree should look for a value to insert or

remove, and also allowing queries. An operation on this structured data should have on average time complexity of log(n), with n being the number of elements of data in the database [42].



CipherText	Plain Text	Hex Value
A4b39424	ape	61 70 65
9f9c299f	bat	62 61 74
9e5e43a8	cat	63 61 74
812c397d	dog	64 6f 67
15575db6	zebra	7a 65 62 72 61

Figure 2.3: B-Tree of the result of a OPE with reference table

2.1.2 Homomorphic Encryption

Homomorphic Encryption comes from homomorphic abstract algebra, that preserves a structure map between the algebraic structures. Basically, Homomorphic Encryption allows some computations to be made in the ciphered text as if they were in plaintext and outputs the result in ciphered text that when deciphered matches the result as if the same operation was made in the original plaintext[56]. There are several drawbacks to Homomorphic Encryption, specially on Fully Homomorphic Encryption, causing it not to be very practical due to its efficiency. Partially Homomorphic Encryption cuts some of the costs of Fully Homomorphic Encryption.
In the following sections we present two types of Homomorphic Encryption.

2.1.2.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption [16] allows for arbitrary computations on encrypted data, namely multiplication and addition operations. Despite the extensive work done in this subject, Fully Homomorphic Encryption continues to not be considered efficient enough for everyday use. One example of this, is the work made in [17], a Fully Homomorphic Encryption work was accomplished with an AES Circuit, but it takes 7 hours to make the first successful operation (despite having the algorithm become faster as more rounds are made) and it also requires a machine with 256 GB of RAM to run it.

2.1.2.2 Partially Homomorphic Encryption

Partially Homomorphic Encryption is a form of Homomorphic Encryption, only certain mathematical homomorphic operations can be made, like additive or multiplicative homomorphism, but not both operations [38]. A Partially Homorphomic Encryption supports adding Homomorphism if and only if: $\xi(x)+\xi(y)=\xi(x+y)$. A Partially Homorphomic Encryption supports Multiplicative Homomorphism if and only if: $\xi(x)*\xi(y)=\xi(x*y)$. Several algorithms are presented below that preserve one of the two mathematical operations available on Partially Homorphomic Encryption:

- **RSA**

  RSA, or more specifically unpadded RSA, is used to exhibit **multiplicative homomorphism**. It multiplies two RSA ciphertexts values, with the decrypted result being the same as if the multiplication was made with the two values in plaintext[16].
  Being the modulus m and exponent e constant integers, the homomorphic property is given by:

$$\xi(x)*\xi(y)=x^e*y^e \mod m=(x*y)^e \mod m=\xi(x*y) \tag{2.1}$$

- **ElGamal**

  Similarly to RSA, ElGamal is also used to exhibit **multiplicative homomorphism**, which means multiplying various elements of a ciphered text and decrypting the result equals to multiplying the values in their original plaintext in unencrypted form [34]. The ElGamal algorithm exhibits multiplicative homomorphism working this way:
  Given a plain text x1, private key a, public keys k1, k2 and k3 and a nonce no1 where $\xi(x1,no1)=(y1,y2)$, where y1 and y2 mean:

$$\begin{aligned} y1 &\equiv k1^{no1} \mod k3 \\ y2 &\equiv x1*k2^{no1} \mod k3 \end{aligned} \tag{2.2}$$

  Multiplying the plaintexts x1 and x2:

$$\begin{aligned} x1*x2 &= (y1,y2)*(y3,y4) \\ &= (y1*y3, y2*y4) \\ &= (k1^{no1}*k1^{no2}, (x1*k2^{no1})*(x2*k2^{no2})) \\ &= (k1^{no1+no2}, x1*x2*k2^{no1+no2}) \end{aligned} \tag{2.3}$$

Decrypting will give us:

$$decrypt(k1^{no1+no2}, x1 * x2 * k2^{no1+no2}) = x1 * x2 \tag{2.4}$$

- **Paillier**

  Paillier is used to exhibit **additive homomorphism**. Each component of multiple ciphertext is multiplied with its respective component. The result when decrypted is equal to the sum of the values in plaintext[39]. As the original paper states, the Encryption and Decryption in this algorithm is made as it is shown bellow.

  Encryption:

  plaintext $m < n$

  select a random $r < n$

  ciphertext $c = g^m * r^n \mod n^2$

  Decryption:

  ciphertext $c < n^2$

  plaintext m $= \frac{L(c^\gamma \mod n^2)}{L(g^\gamma \mod n^2)} \mod n$

  As stated previously the result of multiplying two ciphertexts values will decrypt to the sum of those two values in plaintext, given that x is the first plaintext and y is the second one:

$$decrypt(encrypted(x, r1) * encrypted(y, r2) \mod n^2) = x + y \mod n \tag{2.5}$$

  Pailler encryptions also allows the use of homomorphic multiplication of ciphertext with plaintext.

2.1.3 Computations on Trusted Hardware

Computations on Trusted Hardware is about developing trusted technologies that guarantee the user safety while running the software on theirs devices. A device can be considered trusted if it always behaves in an expected manner even if an adversary tries to attack it [32]. In this section we present several technologies that attempt to execute software on a remote computer owned and maintained by an untrusted party, with some integrity and confidentiality guarantees [12]. In **Figure 2.4** the user relies on a Remote Computer, owned by an untrusted party, to perform some computation on. The user trusts the manufacturer that has a piece of hardware in the remote computer, so the user has assurance of the computation and confidentiality of its data.

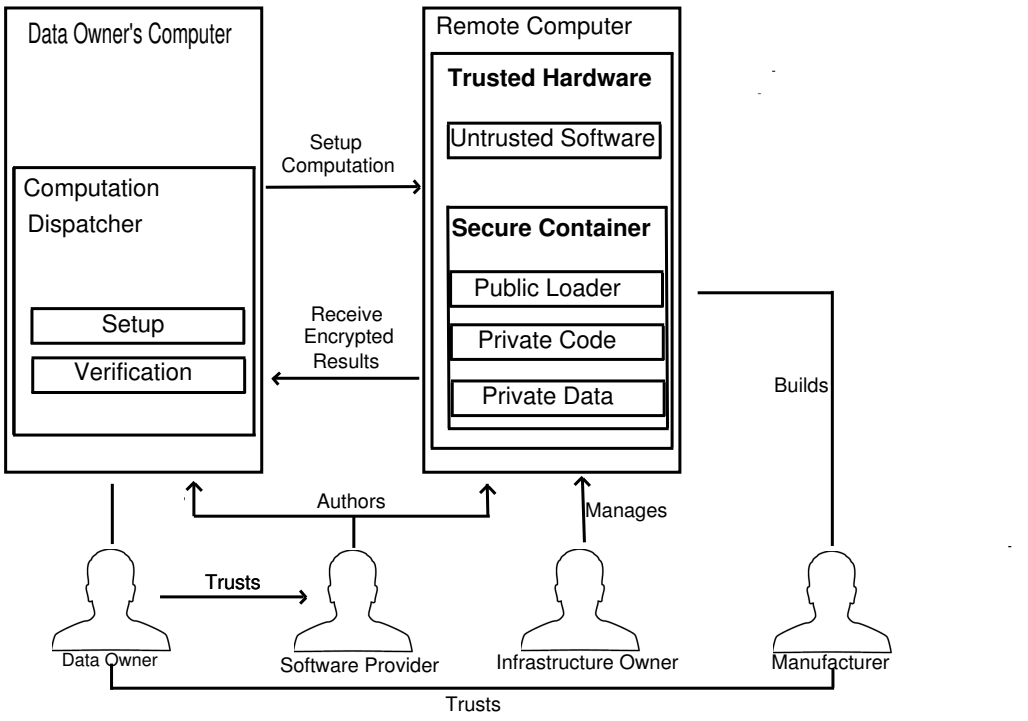


Figure 2.4: Trust Computing on Secure Remote Computation [32]

2.1.3.1 TPM

TPM refers to Trusted Platform Module and is a hardware module incorporated and deployed in a motherboard, smart card or processor that provides the resources needed for trusted computing [49].

Through secret sharing and cooperation with other hardware and software components, the TPM provisions three services: authenticated boot, certification, and encryption.

The authenticated boot service allows to verify that the boot of the entire operating system is made in well defined stages at which only approved versions of the modules of the OS are loaded. This could be done by verifying a digital signature associated with each module. Additionally, a tamper-proof log of the loading process is kept by the TPM that later can be consulted. Tampering is detected using cryptographic hash functions. It is also possible to configure the TPM to include additional hardware and application and utility software in its TCB given some restrictions to prevent threats. This service can be used to guarantee that the machine which hosts the TPM is in a well defined and trusted state after booting.

Despite the advances in TPM, this technology seems to have been compromised in 2010 by Christopher Tarnovsky, but this was disregarded by the manufacturing

company of the chips, Infineon, as the company stated that it was necessary a high skill level of hardware experience, as it required removal of the chip's case top layer, then tapping into a data bus to get the unencrypted data [50]. In October 2017 a new code library by Infineon, exposed some vulnerabilities with the system, as it allowed some private keys to be guessed from the public keys, subsequently a lot of data was compromised [20].

2.1.3.2 Intel® SGX

Intel®SGX (Intel's Software Guard Extensions) is a set of CPU instructions that can be used by applications to set aside private regions of code and data called enclaves [23]. SGX was introduced with the Skylake architecture and is considered the successor to TPM. In SGX, the hardware establishes a secure container and then the user uploads its data and desired computation to the secure container. SGX protects the data's confidentiality and integrity while the computation on the data is being executed [32], even from a potentially malicious OS/Hypervisor.

To be more specific, SGX reserves a space in memory called PRM (Processor Reserved Memory). The PRM contains an EPC (Enclave Page Cache). The EPC contains memory for all enclaves and consist of 4KB pages that store the enclaves data and code, with the maximum of 128MB (32768 pages). The EPC is always encrypted.

The CPU then protects this part of the memory from all non-enclaved attempts to access the memory blocks, this including the kernel, hypervisors and SMM accesses. The initial code and data is uploaded by an the untrusted system software, relying in software attestation, so it can be proved that is the user that is communicating and the the user is interacting with the right enclave. The communication is proofed by a cryptographic signature that certifies the hash of the secure container contents matches with the ones of the expected. After this, the CPU uploads the unprotected memory to EPC pages and commits them to the enclave area. The CPU now marks the enclave to initialized and its contents are hashed. Operations on the enclave part can only undergo under specific CPU call instructions. After all the computation is done, the CPU hashes the results and returns it to the system software.

With Intel SGX version 2 it will be added the ability to dynamically add pages and threads to a running enclave and the feature to increase the enclave's heap during runtime [12].

2.1.3.3 ARM TrustZone

ARM's Trustzone approach to secure computing is to separate secure and non-secure hardware, so that it can stop non-secure software from accessing secure resources directly. For the processor there are 2 types of software: secure and non-secure. It separates the two by having a System-on-Chip (SoC) that virtualizes 2 different CPU's. This way the important assets can be protected from adversary attacks such as software attacks and common hardware attacks [31].
This technology can be implemented in ARM Cortex-A, Cortex-M23 and Cortex-M33 based systems. An overview of the ARM TrustZone can be seen in **Figure 2.5**, where there 2 are distinct hardware CPU virtualizations, one is non-trusted and the other trusted.

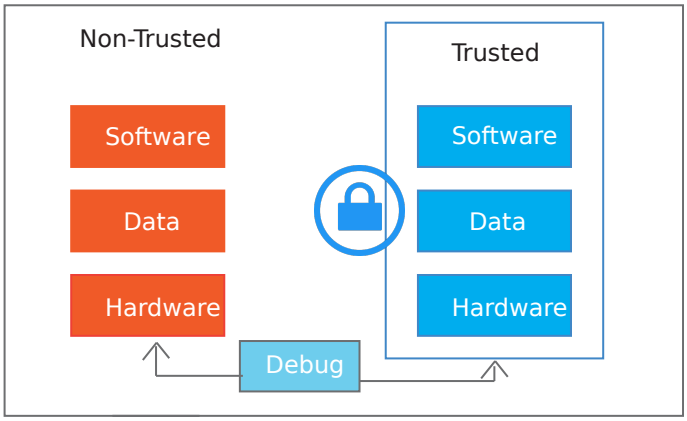


Figure 2.5: Overview of ARM Trustzone [31]

2.2 Searchable Encryption

Searchable Encryption allows the user to outsource a collection of encrypted data on a remote server in a private manner, while still maintaining the ability to make keywords searches over it. These search functionalities are supported without decrypting the data, with the smallest possible loss of confidentiality.

More specifically, Searchable Encryption allows a user to create a search token from the keywords that he wants to search for, in a way that given the token, the server can retrieve the encrypted contents relevant to those keywords. The search token represents the encrypted query and can only be generated by the users with the appropriate secret key [21].

Nonetheless, it includes some type of Deterministic Encryption as the use of deterministic primitives. A disadvantage of this method is the leakage of search and access patterns when some operations are made that adversaries can recognize. For

this reason the security of query keywords and search results should be guaranteed [39].

Searchable Encryption includes 4 parties: the data owner, the semi-trusted server, a collection of users that will read the data and the key generator.

- **The Data Owner** - The data owner outsources all of his data together with appropriate keywords. The user is responsible for encrypting the data with a particular cryptographic scheme like creating an index that enables searching over his encrypted data, see figure 2.6.
- **The Cloud Server** - It's used to outsource the encrypted data. Upon receiving a trapdoor from a user with keywords, the server searches over the encrypted data using the Index created by the data owner. After finishing the search for the relevant content, the server returns it in an encrypted form.
- **The Data User** - The data user that wants to send a query to the server, so he gathers his secret key and creates an encrypted trapdoor containing his query keywords and sends the trapdoor to the server. After completed, the server sends the encrypted results to the user, see figure 2.6. This user can be the same as the data owner.
- **Key Generator scheme** - This party is considered trusted and is responsible for the generation and management of secret keys, like the encryption and decryption keys [44] [53]. In some application, this part can be considered to be the data owner.

In figure 2.6 execute an upload operation, the user builds an index I, based on the Database DB and the collection of Messages W, and sends to the Server the index I and the collection of the encrypted messages, from M1 to Mn. To make a query operation, the user creates a Trapdoor T, containing his secret key and the keywords f, and sends to the server the Trapdoor T. The server then searches over with the index I and the trapdoor T, the relevant messages M and returns them to the user.

There are two main types of Searchable Encryption, those types are Symmetric Searchable Encryption (SSE) and Public-Key Searchable Encryption (PKAS). In the following section we are only going to present Searchable Symmetric Encryption.

2.2.1 Searchable Symmetric Encryption

Searchable Symmetric Encryption (SSE) is one of the two main types of Searchable Encryption, with the other being Public key encryption with keyword search (PEKS).

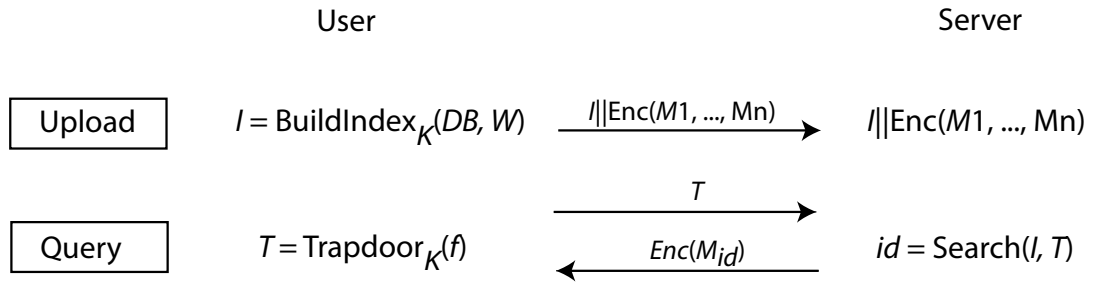


Figure 2.6: Upload and Query scheme in Searchable Encryption [21]

As the name suggests Searchable Symmetric Encryption is associated to symmetric key primitives, as it allows only the private key holder to upload encrypted data and to create trapdoors for queries. On the contrary, Public key encryption with keyword search, is associated to public key primitives. It enables all the users who have the public key to upload encrypted data but only allows the private key holder to create trapdoors for queries [53]. For these disputed reasons SSE is considered more efficient than PEKS [13].

In **figure 2.7** [44] we can see the system model of a Searchable Symmetric Encryption Scheme. The user encrypts a set of data and creates an encrypted index file witch contains a set of encrypted keywords, extracted from the data. The user outsources the index and the encrypted data to the server. During a search the user creates the encrypted query and sends the query to the server. The Cloud server takes this information and retrieves pointers to the document that contain the search keywords in the query and returns it to the client.

2.2.1.1 SSE Schemes

In this subsection we are going to present a number of SSE Schemes.

- **Song**
  This scheme was the first Searchable Symmetric Encryption scheme [48], and it was presented in the year 2000. It consists in encrypting the document's keyword and XORing the encryption with a HMAC of a random generated value. This scheme didn't contain an index, so for searching for a query keyword consisted on verifying the HMAC signatures, meaning that this method consists on having linear search complexity for the total number of the query's keywords[15].

- **Goh**
  Goh's implementation of a SSE scheme [18] consists of having an index for every document, as it was the first scheme containing one. This results on every

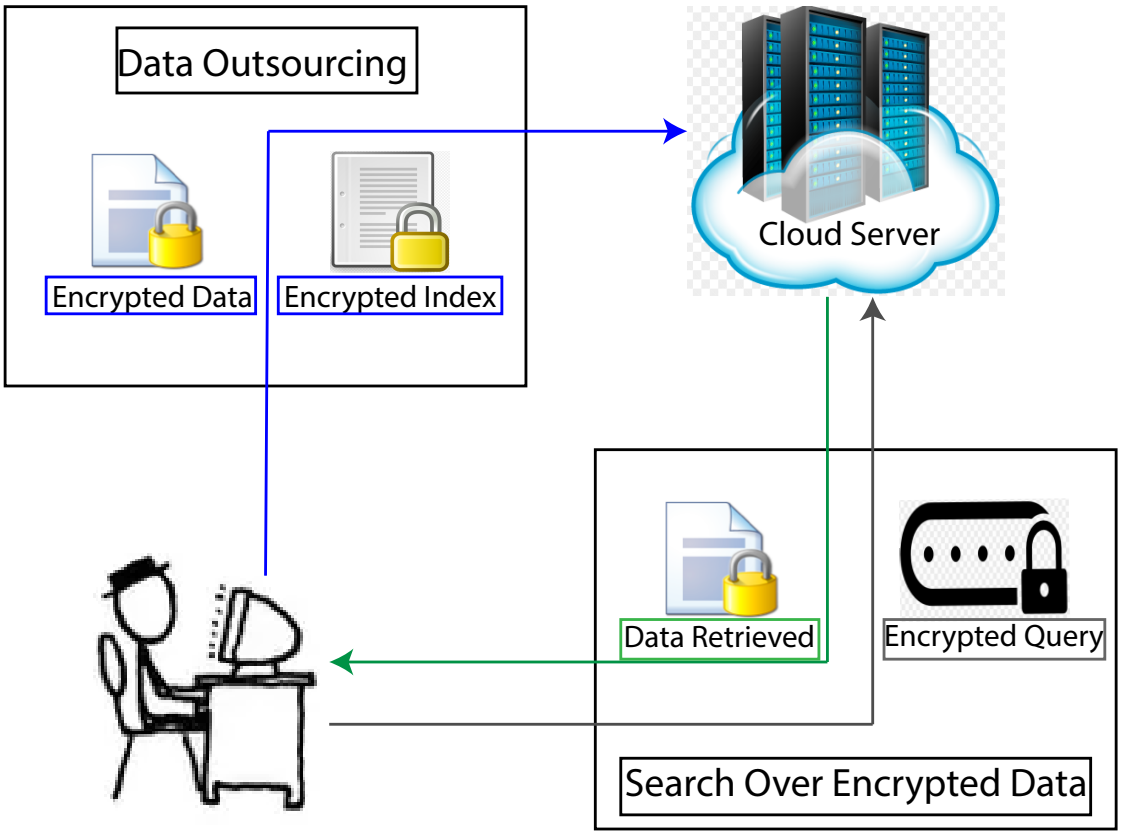


Figure 2.7: System Model of a SSE scheme [44]

time a query is made the server has to search every index for the keywords present in the query, and the amount of computation work that has to be done for a query is proportional to the number of documents in the collection.

- **Oblivious RAMs**
  With Oblivious RAMs, SSE can be fully achieved. Using this technique any type of search query can be done, including disjunctions and conjunctions of words, without showing its access patterns, that means the information is not leaked to the server. But there's a main disadvantage in this method, as it requires multiple rounds for write and read operations and there is a logarithmic cost in the average operation. O. Goldreich and R. Ostrovsky wrote on [19] and a two round solution to this problem, but causing an extreme large square root overhead in the process.

- **SSE-1**
  SSE-1 was presented in [13] and is called a non-adaptively secure construction. Like most SSE schemes all the documents are encrypted and an index I is built during the process of setup.

The index I consists of two different data structures, an array A that for all the distinct words in all the documents, contains a stored encryption of the documents, and a look-up table T, that for all the distinct words in all the documents, contains the information to locate and decrypt the appropriate elements inside data structure A. Despite the efficiency of this algorithm, SSE-1 is not considered secured to adaptive adversaries.

- **SSE-2**
  SSE-2, on the contrary of SSE-1, can achieve security against adaptive adversaries, although trading security for performance, as this scheme takes more space to perform a query and size in the server to store data. However, the computation costs for SSE-2 are the same as in SSE-1.

  SSE-2 is constructed similar to SSE-1, but instead a index I is constructed using the pair values with <Key, Value>, where key is a distinct word from the all the words in all the documents and the value is a pointer to the document where that word appears. Given a word W, that word must be concatenated with a integer J that represent how many times that word has appeared in the set of documents until that point, for example, if the word coin appears 3 times in the set of documents, we are going to have a set of pairs in the Index like <coin1, address1>, <coin2, address2> and <coin3, address3>.

  It's stated in [13] that this method of Indexing will allow the simulator to construct an index for the adversary that is indistinguishable from a real index, therefore, secure to adaptability from the adversary.

- **Cash**
  Cash's implementation of a SSE scheme [9] consists in having a Dynamic Searchable Encryption in Very-Large Databases, where the entries scale to millions of records. It is the fastest of presented schemes if there are a large amount of entries stored in the system. It subsists in associating with each record/keyword a pseudorandom label, and then for each pair storing the encrypted record identifier with that label in a non secure dictionary. The labels are then derived, so when the user makes a keyword query, the client computes some keyword specific short key that allows the server to search by computing the label, and then retrieve the identifiers from the dictionary, and also retrieves the encrypted matching record identifiers. A disadvantage of this method is that the size of the dictionary is compromised to the server, so it can know the number of records, keywords and pairs in the data.

The labels are derived so that the client, on the user inputting the keywords to a query, can compute a keyword-specific short key, allowing the server to search by first recomputing the labels, then retrieving the encrypted identifiers from the dictionary, and finally decrypting the matching encrypted record identifiers.

Table 2.1: Different SSE Schemes [44] [13]

Scheme	Time Complexity	Index Size	Server Storage	#Rounds
Song [48]	$O(n)$	N/A	$O(n)$	1
Goh [18]	$O(n)$	$O(n)$	$O(n)$	1
Oblivious RAMs [19]	$O(log^3(n))$	N/A	$O(n * log^2(n))$	$O(log(n))$
SSE-1 [13]	$O(r)$	$O(m+n)$	$O(n)$	1
SSE-2 [13]	$O(r)$	$O(m*n)$	$O(n)$	1
Cash [9]	$O(r/p)$	$O(n)$	$O(n)$	1

In **table 2.1** we show several computations costs of five different SSE schemes. The letter n denotes the size of the documents set, the letter r denotes the number of documents, the letter p denotes the number of processors and the letter m denotes the size of the keywords. To make the comparison easier and more fair we assume each document has the same size.

2.2.2 Range Queries

The result of a range query operation is all the documents between between a certain range in a database. Using an ordinary Searchable Encryption if we'd like all the results on the interval of [a,b], the user would have to make b - a + 1 operations, causing it to be a very inefficient and insecure method [25].

In [46] it's built a Multi-dimensional range query over encrypted data, where it was adapted a tree structure and developed a multi-dimensional range query system, which included working with an authority holding a master key that can issue searches to an authorized party, allowing it to decrypt data entries whose attributes fall within specific ranges, while still preserving privacy of other data entries.

In [6] it was presented a solution which involved bilinear maps defined over elliptic curves, permitting Conjunctive, Subset, and Range Queries on Encrypted Data, but still depending on a public key scheme technique.

In [25] a method was presented which consisted in utilizing only symmetric key encryption systems, with the search time depending with the number of documents

corresponding to the retrieved documents corresponding to the keywords, and not the entire database. This method involved doing the search query on a linked chain structure, and not in a tree structure. For every entry it is created an external link containing the following node. This way when the user creates a trapdoor for a query in the interval [a,b] it must contain the decryption key for the chain a and the decryption key for chain b By searching two linked chains and all documents related to the interval [a,b] are searched.

2.3 Password Databases

Password Databases are storage engines where the user's credential, including the username and its password are stored. Some examples of how the passwords are stored in modern days system include Plain Text Passwords, Basic Password Hashing, Slow Hashes, Encrypted Passwords and Encrypted Passwords with a Dash of Salt.

This section is divided in two subsections, the first one focuses on how passwords can be stored in a database, stating how each process works and its advantages and disadvantages. In the second subsection we focus on attacks made by an adversary on these password databases.

2.3.1 Password Database Storage in The Industry

Information in password databases should be considered more sensitive, because it contains all the information necessary for an adversary to enter with users stolen password credentials. Most people recycle passwords between websites, so the adversary can easily enter any website with the stolen password.

2.3.1.1 Plain Text Passwords

Plain Text Passwords is the most simple way to store a password and also the most unsafe. The password is stored as it is, and everyone taking a quick glance at the database can see it. This is a very bad idea for companies to store a password, as the password can be checked by an employee, social engineering methods, a backdoor created by a missing Operating System patch or a virus... Hackers can use a simple SQL Injection method to see everyone passwords. This is the equivalent of someone breaking into one's house and finding the key for the safe just on top of it.

2.3.1.2 Password Hashing

Password Hashing consists of generating a String of characters from a given password. The generated String is of a fixed length and based on the possible variations from the inputted password. The algorithms that use hashing are one way functions and are made in a manner that is impossible to obtain the original password. Some examples of algorithms that use Basic Password Hashing are:

- **MD5**
  Originally designed to be a hash function, MD5 has been proven to be a not so secure method of doing so after some vulnerabilities about it being exposed. It is nowadays more often used has a checksum to check data validity and integrity [52] .

- **SHA-3**
  SHA-3 refers to Secure Hash Algorithm 3 is the latest iteration of the former SHA-0, SHA-1 and SHA-2. It was published in just August of 2015 and is based on the KECCAK algorithm [14]. It consists of a set of hash functions and it can only be implemented in a 64-bit processor. It has been proven that it is more secure then its predecessor(SHA-2), but it takes as twice as much time.

In figure 2.8 we see an illustration of hashing a password. Once the password is hashed there's no way to get the password back.

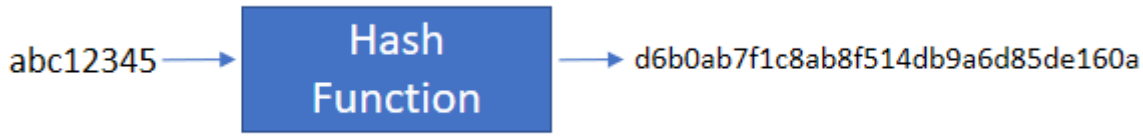


Figure 2.8: Hashing the Password 'abc12345'

2.3.1.3 Password Hashing with salt

Password Hashing with Salt refers to not just hashing the password, but hashing it with the combination of the password + salt. A salt is randomized hash, and it is appended to the password in the beginning of it or the end [14]. This way lookup table and rainbow table attacks will not work, as those tables are only successful if the hashed passwords are the same. We can see an example of password hashing with salt below:

MD5Hash('abc123') = a90acb735b18a3314f2db44104bea194
MD5Hash('abc123' + '9oGQSS1cTd') = 2e8cfc279e0f90e392d0068af2a01f6f
MD5Hash('abc123' + 'K89xywW5PE') = 3a6f3a484e0d39f09760ef03cd70bf2d

The salt used shouldn't be the same for each hash, as the adversaries can still run a dictionary attack to discover the sites 'secret' salt. Instead a salt should be generated every time a new password is inserted into storage. Also, the salt shouldn't be short in terms of length, as it can be easily discovered by adversaries using a lookup table.

- **Bcrypt**
  Bcrypt is a password hashing scheme [33], based on the Blowfish block cipher, that makes use of the salt, making it very useful against rainbow table attacks. Bcrypt takes as input the cost, the salt and the password to be hashed. The input cost determines how expensive this function is going to be, as this is an adaptive function, the number of iterations in bcrypt are directly related to the value of the input cost, making the algorithm slower for every new iteration, although, theoretically meaning the final hash will become more secure. The more processing power this algorithm takes to hash, the more secure it will be. This algorithm is considered future-proof, because as the processors become more powerful and cheap over time, brute force attacks will become easier, but so will the bcrypt become more secure as its security depends on the processing power involved in the process of hashing. For every iteration of this algorithm the generated key gets expanded, with the use of the original password and the original salt, making it more difficult to crack, like this. The returned result of this algorithm is a bcrypt hash [33].

2.3.1.4 Encrypted Passwords

Encryption of passwords is the process of transforming the password to a series of characters, that are unreadable to the human eye. As already stated in this work, symmetric encryption allows a party with access to the key to decrypt the encrypted to obtain the original password. Some examples of some algorithms that use Basic Password Hashing are:

- **RSA**
  RSA refers to Rivest–Shamir–Adleman and is on the first forms of public-key encryption and the standard for transmitting information over the web. This is how your standard Whats App messages and other INTERNET ways of sending over information are usually encrypted. This method is relatively slow, so it just

usually passes the public encryption key around, so the encryption/decryption operations can be executed faster at a much higher speed[43]. As there is no random factor in the process of the RSA algorithm, it is fully deterministic, meaning if an adversary knows the linear relation from the plaintext to the cypertext, can easily crack it [11].

- **AES**
  AES refers to Advanced Encryption Standard and is a symmetric key algorithm used the by USA government to cypher encrypt any sensible data and information. This method has been proven over the time to not be so secure, as various methods have been proofed to break the AES algorithm, more notably, brute force methods [4]. The AES algorithm is considered of very high speed as it doesn't use a lot of computer resources as RAM and CPU, as it only requires 16 cycles per byte of encrypted data [7].

- **PBKDF2**
  PBKDF2 refers to Password-Based Key Derivation Function 2 and applies a pseudo-random function to derive keys. This derived key has no limits in terms of length, although it can be limited to the size of the structure of the pseudo-random function. It is remarkable how hard it is to brute force this algorithm, due to its huge number of layers of encryption (minimum recommended is 4096). The PBKDF2 method takes as input the password, a salt, the number of layers and the desired length of th e resulting key, it outputs the ciphered text. Although it is considered very hard to brute force, it is mentioned that this method is very slow [37].

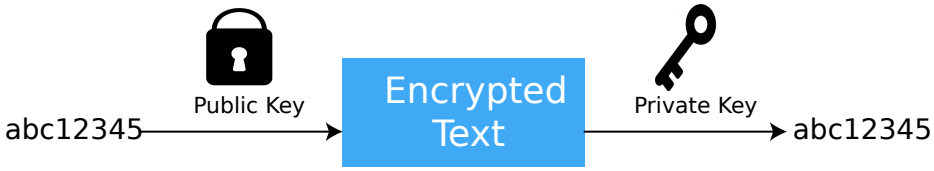


Figure 2.9: Encryption and Decryption of the Password 'abc12345'

In figure 2.9 we see an illustration of encrypting a password. A public key is requires to encrypt the text and the private key is necessary to decrypt it.

2.3.1.5 Encryption vs Hashing

Encryption and Hashing are the main ways of storing passwords in a database, and both operate on the original plaintext password. The main difference between

them is that in hashing once you hash a password there's no way to get back to the original plaintext, on the opposite, encrypting a password permits getting the original password as long as the private key is given. It's disputed that the best way to store a password is to hash it, because, supposedly, the hashed password can't be hashed back, although this can be avoided by adversaries with a dictionary or a rainbow attack. On the other hand if the adversary gains access to the private key, he decrypt the ciphertext he can gain access to all the plaintext passwords.

2.3.2 Attacks on Password Databases

According to a recent study made in the research work [36] this is how the attacks on main websites were made:

1. In the largest percentage of cases (35.3%), the entity that was victim of the security breach chose not to disclose the attack mechanism.

2. **SQL injection** — accounts for 29.4% of the attacks, the most prevalent mechanism for disclosed attacks.

3. **Account hijacking** — (14.7%), wherein access was obtained via stolen account credentials, is the second most prevalent attack methodology.

4. **Spear-phishing and 3rd party software flaws** — were tied at 5.9% each

There's close to nothing we can do about Account Hijacking and Spear-Pishing, the only way to efficiently solve this problem is to force a two-way step login verification. In this situation the users logins using the knowledge factor, something the user knows like his password, and a possession factor, something only the user has, like his mobile phone, per example, if the user wishes to login in a website he has to insert his password, and then use some form of interaction with his phone, like inputting a generated code.
In the case of SQL Injection, or other methods that an adversary takes a snapshot of the database storage, the data comes in plaintext, hashed or encrypted. If the data is in plaintext, the adversary can already see the data he wishes and sensitive information is already considered compromised. If it is the hashed, the adversary has to hash it back to get the original text and if it is encrypted only with the key can he decypher the ciphertext back to its original plaintext.
In this section we are going to present the most common and efficient methods of attacking private information, used by an adversary.

2.3.2.1 Brute force Attack

A brute force attack, also referred as Exhaustive Key Search, consists of an adversary trying to find a password or pass-phrase by trying every single possible combination of characters available. This method may be efficient against short passwords, but not long passwords, as the possible password combination will be much larger [27]. As we can see in **Table 2.2** there are 97 possible combinations (26 + 26 + 10 + 35) for a digit to make up a password character.
A 3 digit password would take at worst 97^3 (912 673) attempts for an adversary to guess the password by using brute force, with the current CPU and GPU available in the market it is cheap and easy to brute force a password with only 3 digits. Currently, the standard minimum size for a password is 8 digits and that would take 97^8 (7 837 433 594 376 961) attempts at worst, it is not a viable option for an adversary to brute force its way with so many possible combinations.

Table 2.2: Possible digits that make up a password character.

Type	Digits	Number
Lowercase letters	*'a' to 'z'*	26
Uppercase letters	*'A' to 'Z'*	26
Numbers	*0-9*	10
Special Characters	*"^!'#$%&'()*+,-./:;<=>?@_ ´'{/}«»~*	35
Total		97

2.3.2.2 Dictionary attack

A Dictionary Attack is very similar to a Brute Force Attack, but instead of trying to force its way with every possible combination, it uses a technique, that only tries words contained in a dictionary. The dictionary is created by an adversary, and it contains all the most likely words to be contained inside a password, therefore reducing the number of attempts on cracking a password. In every attempt, if a word from the dictionary fail, the algorithm tries another word. The appending of a salt in the hash of passwords, or even on the plaintext ones, can totally render the use of a Dictionary attack useless [10], as the salt appends a random string of characters to the passwords that most likely won't exist in the dictionary.

The dictionary can contain real plaintext words or hashed text from the real plaintext words. More advanced dictionary attacks can also attempt to use common special characters, numbers, words and a mixing combination of all of them.

The adversary can use the most relevant keywords available on the user's profile on an attempt to fill the dictionary in a more efficient way. The user can also avoid an dictionary attack by misspelling words on his password, to words that are less probable to don't exist in the dictionary created by the adversary.

2.3.2.3 Birthday Attack

The Birthday Attack uses the same approach as the Birthday Problem from Probabilistic Theory. The Birthday Problem consists on having a population of n random people, and considering some pair of two of those n people will share the same birthday [55].

Using the same logic an adversary, that has gained information about one password, can try to guess that there are at least two passwords equal to one another in the database storage, given that are enough stored passwords.

2.3.2.4 Rainbow table

Considered to be the most efficient way of cracking a password, a Rainbow Table consists on an adversary building a pre-computed table with the same hash algorithm of the hashed password its trying to crack, and a reduction algorithm that's used to reverse a password hash to a plaintext. The chains which make up rainbow tables are of a hash and reduction functions starting at a certain plaintext, and ending at a certain hash. The Rainbow table algorithm starts with a hash and works like this [22]:

1. If the Hash you have is the same as one in the list of final Hashes, go to step 6.
2. Insert the Hash in the list of final Hashes
3. Reduce the hash to another plaintext
4. Hash the reduced plaintext
5. If the hash obtained is the same as the original hash, the intended plaintext is discovered. If not, choose a random plaintext, hash it and go to step 1

This method also takes a long time to run from scratch, but with the difference that the adversary doesn't have to start from zero every time he wants to crack a hashed password, and it can index and search the previously completed hash results [51].

A Rainbow Table does a space-memory trade-off, which means, it trades the long computing time to discover a password, for memory, meaning the longer the method

runs, the more disk space will occupate in the hard drive or RAM. A Rainbow Table size will vary, from 52 GB up to 864 GB,[30] although on the long run it will probably take much less time to be successfully for the adversary, than a Dictionary Attack or a Brute Force Attack [26].

2.3.3 Mitigating Attacks on Password Databases

2.3.3.1 Password Reuse

In [24] it's stated that most password theft happens because of the Domino Affect on Password Reuse, meaning that the users use the same 4 or 5 password for every website. The problem with this is that if an adversary gains access to a password database of just one website, that could lead to failure on other systems.

One way to prevent attacks is when the system detects an adversary repeatedly attempting to use incorrect password to access one's account, the system would shut down in defence.

The paper also mentions public-key encryption (PKE), Public-key infrastructure (PKI) and Biometrics to serve as the authentication process, instead of the usual user-password combination.

Biometrics involve some form of data obtained from ther user's body such as, the iris of his eyes, the fingertip, the whole face, or some patterns in the user's voice. While convenient, unlike a password, once compromised the biometric authentication can't be changed.

2.3.3.2 Password Managers

Password Managers helps users manage multiples password accounts by turning a single memorized password into a different password for every of is accounts. Typical password managers are integrated to web browsers. When the password managers detects that the user has return to a site for which he has a stored password, it fills the login form with the appropriate username and password. Most password managers do a trade off in their design, trading the process of logging in more conveniently, reducing the user's memory burden for every website, but introduces an external dependency [54].

Some usability goals should be guaranteed in password managers like, let the user only have to remember one password, allow the user to change said password and security goals like, the use of an unique password for each site, resist offline attacks and automatically detect pishing sites. This should relief the user of having

to create and to memorize a new password for every website that the user is signed up to.

Although the advancements on password managers over the years some studies made recently like in [29] showed some critical vulnerabilities, and even stating how the attackers could steal arbitrary credentials of the user from the most popular password managers around the world.

2.3.3.3 Reverse Turing Tests

Reverse Turing Tests, also referred as just RTT, and better known by some people for the name CAPTCHAs, are a series of tests that distinguish between a computer program and a human. These RTT should be easy for the real user to solve, hard for automated programs to resolve and have a very low probability of guessing the answers correctly [41]. This is useful because it can limit the number of automated attacks on a database password, so a real human has to pass the test every time he wants to try a user password combination. Some examples of RTT could be asking the user to input the words contained in a picture, a math operation, picking some images, etc...

In figure 2.10 we can see two examples of RTT. In the first one the user is required to select all the images containing a bus, and in the second example the user is required to type the words in that picture.

Figure 2.10: Example 1 [1] and Example 2 [2] of Reverse Turing Tests

2.3.4 Scientific Solutions to Secure Password Databases

2.3.4.1 PolyPasswordHasher

PolyPasswordHasher is a software password storage mechanism deployed on the the server side. The technique used prevents the adversaries to crack an individual password, because PolyPasswordHasher uses cryptographic hashing and threshold

cryptography to combine password hash data with shares so that users unknowingly protect each other's password data [8]. With the amount of increased work on the server by a magnitude order, the adversary needs more time to crack a password.

Basically this scheme additionally protects hash password with salt with an additional secret shared used with the Shamir's Secret Sharing method [45]. The secret shares are stored in memory and are used to verify the hashed passwords, although it's guaranteed that an adversary can only read what's on is persisted on disk, including the password database, but can't read from the memory. A disadvantage of this, is that if an adversary gains a access to a single space of memory it can steal the secret share.

2.3.4.2 ErsatzPasswords

In [3] a scheme is presented similar to traditional password database schemes, when an adversary tries to access the database, the only passwords that he will get, will be fake passwords, regarded in this paper as ErsatzPasswords. Then, once the adversary tries to use these fake passwords to authenticate to a user's account, the system detects it and raises an alarm. The fake passwords should maintain the same format and appearance of typical passwords to succeed in the process of deception on the adversary. The authors of the paper claim it is impossible for an attacker to recover user passwords from the hashed format, without physical access to the systems database where the passwords are stored.

2.4 Summary of the Related Work

As seen in the previous sections of this chapter, the outsourcing of data to an untrusted hardware server poses great danger to the user because of the unknown parties responsible for maintaining them. Methods like Searchable Symmetric Encryption have proven to be an efficient way to make searches over encrypted data showing potential for securing password databases, but despite this operations can still leak patterns that adversaries can exploit. Oblivious RAMs avoids this problem with a dynamic solution, frequently switching the position of indexes, and although it is considered a secure method it comes with the great cost in efficiency.

Recent technologies, like Intel SGX, allow creating trusted execution environments where computations can be performed in isolation from other (possibly priviliged) processes, although there are not any work done in password database management on this technology.

Chapter 3

Approach to the Elaboration Phase

In this chapter we present the initial overview of our system and the architecture of our proposal, as well as our vision for the final product of the thesis. Note, that this is only our initial vision of the final work, and some concepts may change before the delivery of the final report.

3.1 Overview of System Model and Architecture

As already stated, our work in this thesis will focus in the implementation of new mechanisms to protect password databases, based on cryptography and trusted hardware. This section briefly explains the basics of the model approach to the work, and may be subject to change during the rest of the thesis.

The system model is composed of three parties: the user, the web application and the cloud server. An example of the system model of the proposed work to be built is presented in Figure 3.1.

The user would interact with the web application to perform his desired operation, like registering, make a login, etc.. The Web Application will then communicate with the cloud server through a SSL and HTTPS secure connection, transmitting the user's desired operation.

In a first tentative solution, the cloud will be responsible for encrypting the data and store it. Its functions also require to when given the login information (username and password) checking the encrypted data and output the expected result. The information will be stored using deterministic encryption. The password, before encryption, will consist of the concatenation of the plaintext with a designated salt.

If the output is correct the cloud emits a temporary token to the web application, so the user may be signed in. Part of the model is not yet fully defined, like where the keys assigned to encrypt/decrypt the data are going to be stored, how and where the encryption process is going to be done and how the entries are going to be searched over. These details of the solution will be fine tuned in the first weeks of the elaboration phase.

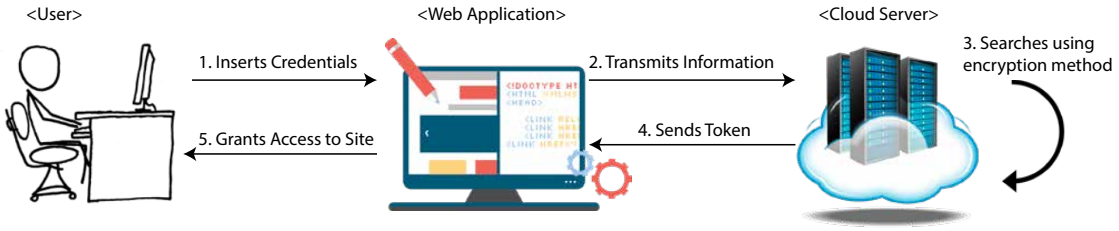


Figure 3.1: System Model to be built in this thesis

3.2 Set of Universal Function Calls

The work in this thesis will also consist in constructing a common API for the password database management systems. This API will be followed by the different solutions we develop in the elaboration phase to provide their functionalities.

The API will feature the expected set of operations that exist in today's password database systems, with the objective of having a universal, clear and well defined set of system calls that will help developers building secure password database management systems. This API will also help us with performance benchmarks and to compare our different solution.

Our trusted hardware scheme will most probably be based on Intel SGX. Since this is a novel work, our implementation will have to start from the ground up. Other schemes, like Searchable Symmetric Encryption schemes, already have implementations available in the Distributed Systems group of the faculty, which will help us implementing these schemes.

Primary API

- Login(username, password)
- addUser(username, password)

Secondary API

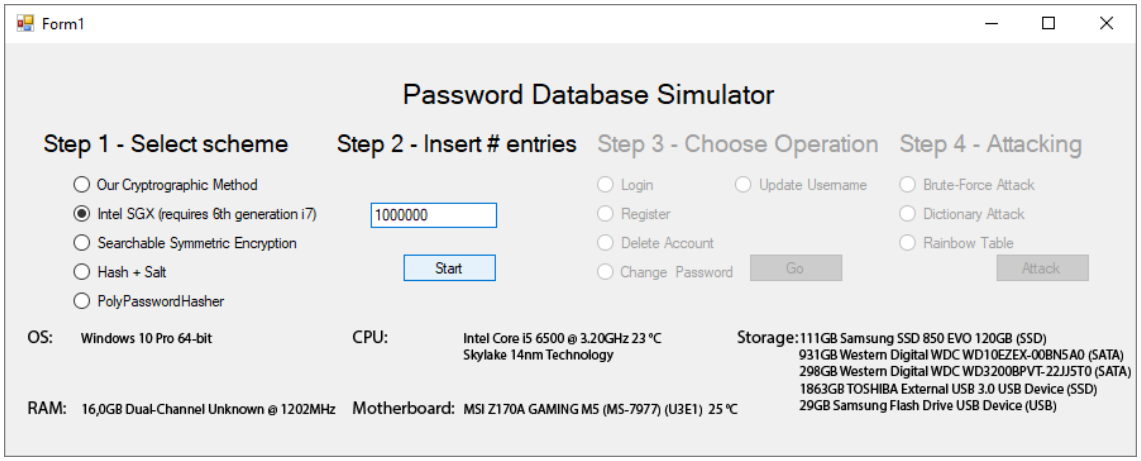


Figure 3.2: Sketch Up of the Graphic Interface

- updatePassword(username, passwordOld, passwordNew)
- updateUsername(username, password, newUsername)
- removeUser(username, password)

3.3 Construction of the Testing Phase

The metrics that we want to measure for the schemes mentioned are performance, scalability and security. To better test and navigate the built components we plan on designing and building a password database management application with graphical interface. This application will feature various steps for constructing a password database for simulation purposes, and will make use of the API mentioned above. It is expected that the application will feature this following steps:

- Step 1, will ask the user which encryption scheme to use.
- Step 2, will ask how many entries does it want to test the database with, being 100 000 the minimum recommended number of password entries, for realistic test benchmarks.
- Step 3, will ask the user to choose a operation, performing benchmarks on the chosen input.
- Step 4, will potentially try to track every possible attack combination to see how much efficient and secure is the database created, that was just built in Step 1 and Step 2.

Figure 3.2 features the sketch up of the initial envisioned application, although the final version can suffer major changes in terms of aesthetics and functionality.

The acquisition of a real world, large data set, containing real user's passwords will be of most importance for the output of true benchmark results purposes.

Chapter 4

Elaboration Plan

In this chapter we present the work plan for the elaboration phase of the thesis, which includes the tasks and milestones that we look forward to achieve.

4.1 Work Plan

The plan of the elaboration of this thesis is divided into 3 major groups.

In the first group, which we call the construction phase, we are going to focus in the creation of our own cryptographic schemes and also the implementation of schemes that already exist, for them all to work with an universal API. The first group in planned to go from the beginning of the thesis until July.

The second group, that we call Evaluation, will focus on the testing of the content we created in group 1. This group is planned to go from the end of group 1, on July until September.

The third group, that we call Writing Documentation, will consist on creating the necessary documentation. It's planned to commence in July and finish on the deadline of the thesis deliver, in September 23th. The work phase is divided into the following tasks:

- **Group 1** - Construction Phase
  - **Task 1** - Refinement of the proposed API and designed solutions
  - **Task 2** - Implementation of secure password database management systems from the literature

  - **Task 3** - Conception and implementation of a novel password database system based on cryptographic primitives
  - **Task 4** - Conception and implementation of a novel password database system based, based on Trusted Hardware, Intel SGX
  - **Task 5** - Development of an application for the demonstration and execution of experimental tests

- **Group 2** - Evaluation
  - **Task 6** - Evaluation and experimental assessment of the work done in task 3
  - **Task 7** - Evaluation and experimental assessment of the work done in task 4
  - **Task 8** - Security analysis, recreation and analysis of common attacks made to password database systems

- **Group 3** - Writing Documentation
  - **Task 9** - Writing and submission of a scientific report for INForum 2018
  - **Task 10** - Writing of the final report

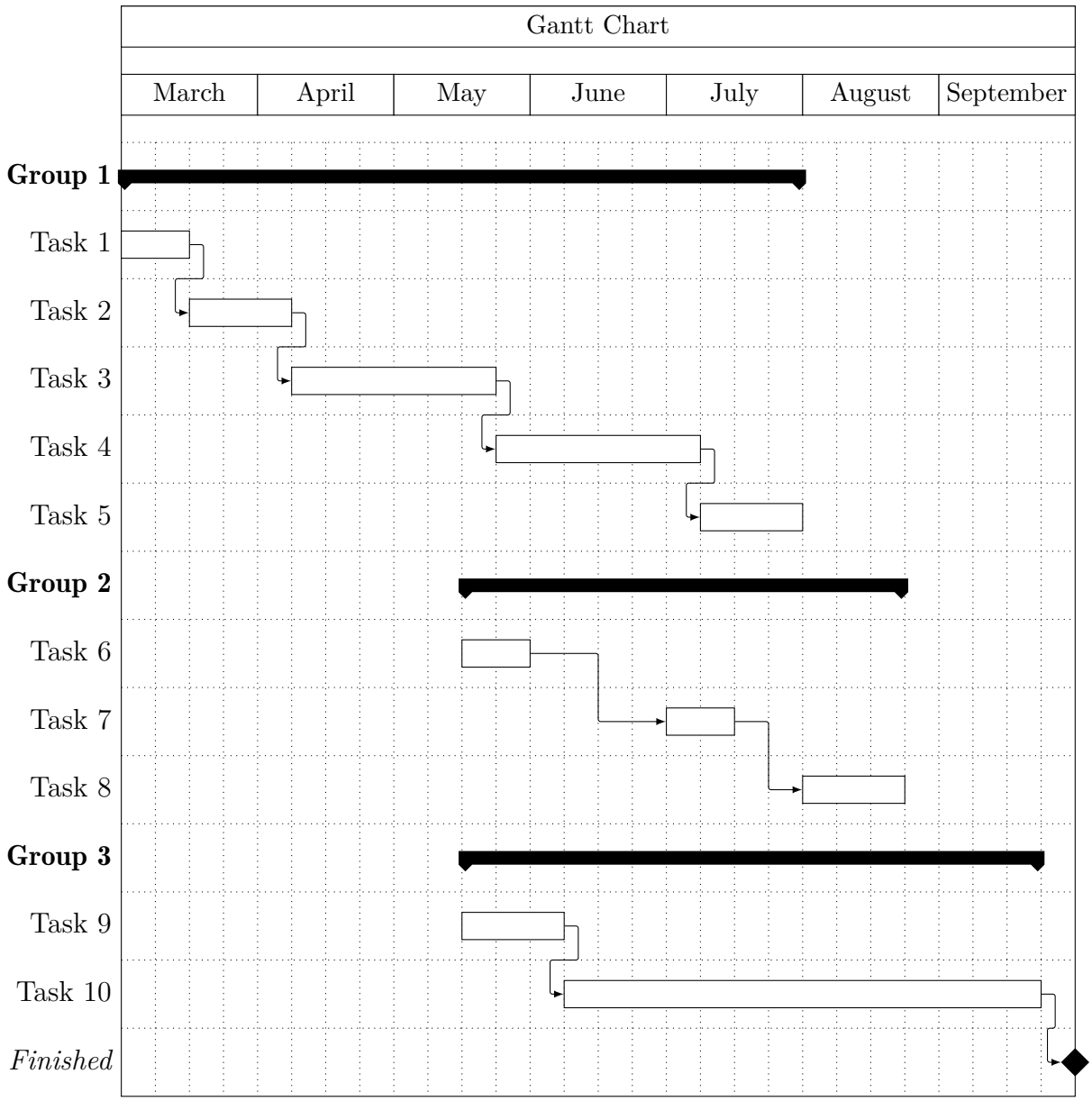


Figure 4.1: Gantt Chart with planned activities for the elaboration phase.

Bibliography

[1] URL: https://www.google.com/recaptcha/api2/demo.

[2] URL: http://www.captcha.net/.

[3] M. H. Almeshekah, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford. "Ersatzpasswords: Ending password cracking and detecting password leakage." In: *Proceedings of the 31st Annual Computer Security Applications Conference*. ACM. 2015, pp. 311–320.

[4] A. Bogdanov, D. Khovratovich, and C. Rechberger. "Biclique Cryptanalysis of the Full AES." In: *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*. 2011, pp. 344–371. DOI: 10.1007/978-3-642-25385-0_19. URL: https://doi.org/10.1007/978-3-642-25385-0_19.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. "Order-Preserving Symmetric Encryption." In: *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*. 2009, pp. 224–241. DOI: 10.1007/978-3-642-01001-9_13. URL: https://doi.org/10.1007/978-3-642-01001-9_13.

[6] D. Boneh and B. Waters. "Conjunctive, subset, and range queries on encrypted data." In: *Theory of Cryptography Conference*. Springer. 2007, pp. 535–554.

[7] D. W.D.W.C. H. Bruce Schneier John Kelsey and N. Ferguson. "Performance Comparison of the AES Submissions." In: 1999.

[8] J. Cappos and S. Torres. *PolyPasswordHasher: Protecting Passwords In The Event Of A Password File Disclosure*. Tech. rep. 2014.

[9] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation." In: *NDSS*. Vol. 14. Citeseer. 2014, pp. 23–26.

[10] S. Chakrabarti and M. Singhal. "Password-Based Authentication: Preventing Dictionary Attacks." In: *IEEE Computer* 40.6 (2007), pp. 68–74. DOI: 10.1109/MC.2007.216. URL: https://doi.org/10.1109/MC.2007.216.

[11] D. Coppersmith. "Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities." In: *J. Cryptology* 10.4 (1997), pp. 233–260. DOI: 10.1007/s001459900030. URL: https://doi.org/10.1007/s001459900030.

[12] V. Costan and S. Devadas. "Intel SGX Explained." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 86.

[13] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. "Searchable symmetric encryption: Improved definitions and efficient constructions." In: *Journal of Computer Security* 19.5 (2011), pp. 895–934. DOI: 10.3233/JCS-2011-0426. URL: https://doi.org/10.3233/JCS-2011-0426.

[14] M. J. Dworkin. "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions." In: *Journal of Object Technology*. 2015.

[15] B. L.d. S. Ferreira. "Privacy-preserving efficient searchable encryption." In: (2016).

[16] C. Gentry. "Fully homomorphic encryption using ideal lattices." In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. 200, pp. 169–178. DOI: 10.1145/1536414.1536440. URL: http://doi.acm.org/10.1145/1536414.1536440.

[17] C. Gentry, S. Halevi, and N. P. Smart. "Homomorphic evaluation of the AES circuit." In: *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 850–867.

[18] E.-J. Goh et al. "Secure indexes." In: *IACR Cryptology ePrint Archive* 2003 (2003), p. 216.

[19] O. Goldreich and R. Ostrovsky. "Software Protection and Simulation on Oblivious RAMs." In: *J. ACM* 43.3 (1996), pp. 431–473. DOI: 10.1145/233551.233553. URL: http://doi.acm.org/10.1145/233551.233553.

[20] D. Goodin. *Millions of high-security crypto keys crippled by newly discovered flaw*. 2017. URL: https://arstechnica.com/information-technology/2017/10/crypto-failure-cripples-millions-of-high-security-keys-750k-estonian-ids/.

[21] F. Hao, X. Yi, and E. Bertino. "Editorial of special issue on security and privacy in cloud computing." In: *J. Inf. Sec. Appl.* 27-28 (2016), pp. 1–2. DOI: 10.1016/j.jisa.2016.04.003. URL: https://doi.org/10.1016/j.jisa.2016.04.003.

[22] *How Rainbow Tables work*. URL: http://kestas.kuliukas.com/RainbowTables/ (visited on 01/16/2018).

[23] *intel sgx for dummies*. 2015. URL: https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx.

[24] B. Ives, K. R. Walsh, and H. Schneider. "The domino effect of password reuse." In: *Communications of the ACM* 47.4 (2004), pp. 75–78.

[25] N.-S. Jho, K.-Y. Chang, D. Hong, and C. Seo. "Symmetric searchable encryption with efficient range query using multi-layered linked chains." In: *The Journal of Supercomputing* 72.11 (2016), pp. 4233–4246.

[26] M. Kalenderi, D. N. Pnevmatikatos, I. Papaefstathiou, and C. Manifavas. "Breaking the GSM A5/1 cryptography algorithm with rainbow tables and high-end FPGAS." In: *22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, August 29-31, 2012*. 2012, pp. 747–753. DOI: 10.1109/FPL.2012.6339146. URL: https://doi.org/10.1109/FPL.2012.6339146.

[27] L. R. Knudsen and M. Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011. ISBN: 978-3-642-17341-7. DOI: 10.1007/978-3-642-17342-4. URL: https://doi.org/10.1007/978-3-642-17342-4.

[28] S. Larson. *Every single Yahoo account was hacked - 3 billion in all*. 2017. URL: http://money.cnn.com/2017/10/03/technology/business/yahoo-breach-3-billion-accounts/index.html.

[29] Z. Li, W. He, D. Akhawe, and D. Song. "The Emperor's New Password Manager: Security Analysis of Web-based Password Managers." In: *USENIX Security Symposium*. 2014, pp. 465–479.

[30] *List of Rainbow Tables*. 2007. URL: http://project-rainbowcrack.com/table.htm (visited on 01/15/2018).

[31] A. Ltd. *TrustZone – Arm*. URL: https://www.arm.com/products/security-on-arm/trustzone.

[32] P. Maene, J. Gotzfried, R. De Clercq, T. Muller, F. Freiling, and I. Verbauwhede. "Hardware-Based Trusted Computing Architectures for Isolation and Attestation." In: *IEEE Transactions on Computers* (2017).

[33] K. Malvoni and J. Knezovi. "Are your passwords safe: Energy-efficient bcrypt cracking with low-cost parallel hardware." In: *WOOT'14 8th Usenix Workshop on Offensive Technologies Proceedings 23rd USENIX Security Symposium*.

[34] A. V. Meier. "The ElGamal Cryptosystem." In: 2005.

[35] P. Mell, T. Grance, et al. "The NIST definition of cloud computing." In: (2011).

[36] D. Mirante and J. Cappos. "Understanding Password Database Compromises." In: 2013.

[37] K. M. Moriarty, B. Kaliski, and A. Rusch. "PKCS 5: Password-Based Cryptography Specification Version 2.1." In: *RFC* 8018 (2017), pp. 1–40. DOI: 10.17487/RFC8018. URL: https://doi.org/10.17487/RFC8018.

[38] L. Morris. "Analysis of Partially and Fully Homomorphic Encryption." In: Rochester Institute of Technology, Rochester, New York, 2013.

[39] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes." In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16. URL: https://doi.org/10.1007/3-540-48910-X_16.

[40] O. Pandey and Y. Rouselakis. "Property preserving symmetric encryption." In: *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*. Springer-Verlag. 2012, pp. 375–391.

[41] B. Pinkas and T. Sander. "Securing passwords against dictionary attacks." In: *Proceedings of the 9th ACM conference on Computer and communications security*. ACM. 2002, pp. 161–170.

[42] R. S.Y. X. Rakesh Agrawal Jerry Kiernan. "Order Preserving Encryption for Numeric Data." In: SIGMOD '04, 2013, pp. 563–574.

[43] R. L. Rivest, A. Shamir, and L. M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems (Reprint)." In: *Commun. ACM* 26.1 (1983), pp. 96–99. DOI: 10.1145/357980.358017. URL: http://doi.acm.org/10.1145/357980.358017.

[44] M. I. Salam, W.-C. Yau, J.-J. Chin, S.-H. Heng, H.-C. Ling, R. C. Phan, G. S. Poh, S.-Y. Tan, and W.-S. Yap. "Implementation of searchable symmetric encryption for privacy-preserving keyword search on cloud storage." In: *Human-centric Computing and Information Sciences* 5.1 (2015), pp. 1–16.

[45] A. Shamir. "How to share a secret." In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[46] E. Shi, J. Bethencourt, T. H. Chan, D. Song, and A. Perrig. "Multi-dimensional range query over encrypted data." In: *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE. 2007, pp. 350–364.

[47] M. Snider and E. Weise. *500 million Yahoo accounts breached*. USA TODAY. URL: https://www.usatoday.com/story/tech/2016/09/22/report-yahoo-may-confirm-massive-data-breach/90824934/.

[48] D. X. Song, D. Wagner, and A. Perrig. "Practical techniques for searches on encrypted data." In: *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE. 2000, pp. 44–55.

[49] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee. *Computer security: principles and practice*. Pearson Education, 2012.

[50] M. Szczys. *TPM crytography cracked*. 2010. URL: https://hackaday.com/2010/02/09/tpm-crytography-cracked/.

[51] *UNDERSTANDING RAINBOW TABLES*. 2016. URL: https://www.drchaos.com/understanding-rainbow-tables/ (visited on 01/15/2018).

[52] X. Wang and H. Yu. "How to Break." In:

[53] Y. Wang, J. Wang, and X. Chen. "Secure searchable encryption: a survey." In: *Journal of communications and information networks* 1.4 (2016), pp. 52–65.

[54] K.-P. Yee and K. Sitaker. "Passpet: convenient password management and phishing protection." In: *Proceedings of the second symposium on Usable privacy and security*. ACM. 2006, pp. 32–43.

[55] M. Yung, ed. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002. ISBN: 3-540-44050-X. DOI: 10.1007/3-540-45708-9. URL: https://doi.org/10.1007/3-540-45708-9.

[56] L. Zhang, Y. Zheng, and R. Kantola. "A Review of Homomorphic Encryption and its Applications." In: *Proceedings of the 9th EAI International Conference on Mobile Multimedia Communications, MobiMedia 2016, Xi'an, China, June 18-20, 2016*. 2016, pp. 97–106. URL: http://dl.acm.org/citation.cfm?id=3021405.