UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

Trabajo Final de Grado (TFG)

Escuela tecnica superior de Ingenieros de Telecomunicacion

Implementing and characterising cryptographic schemes for the Cloud through a Chrome Extension

Aitor Alvar Navarro Iranzo

Tutor: Luis Guijarro Coloma

Abstract

We analyze the importance of privacy preservation in Cloud environments. Mention tools that help the user to remain in control of her data, with special focus on encryption. We also develop a Chrome Extension for analyzing different cryptographic schemes, implementing some of the most used ones on privacy preserving DB management systems (or Protected Search DBMS), simulating such systems through the transfer of the encrypted data to a back-end hosted in Amazon Web Services *(AWS)*. We also provide an in-program characterization of the used scheme.

Resumen

En este proyecto analizamos la importancia de la preservación de la privacidad en entornos Cloud. Mencionamos las herramientas que ayudan al usuario a mantener el control sobre la confidencialidad de sus datos, haciendo especial hincapié en la encriptación. También desarrollamos una extensión de Chrome que permite analizar distintos esquemas criptográficos, implementando algunos de los más usados en sistemas de manejo de bases de datos que preservan la privacidad (Protected Search DBMS). Simulamos dichos sistemas mediante la transmisión de los datos encriptados a un back-end alojado en una instancia en *AWS*. Por último, en la extensión proveemos una caracterización del esquema empleado.

Resum

En aquest projecte analitzem la importància de la preservació de la privacitat en entorns Cloud. Mencionem les eines que ajuden a l'usuari a mantindre el control de les seues dades, amb especial atenció a l'encriptació. També desenvolupem una extensió de chrome que permet analitzar distints esquemes criptogràfics, implementant i caracteritzant alguns dels més utilitzats a sistemes de maneig de bases de dades que preserven la privacitat (Protected Search DBMS), simulant aquests sistemes mitjançant la transmisió de les dades encriptades a un back-end allotjat a *AWS*.

Motivation

Data is the new oil.

In recent years experts and institutions have tried to convey the growing relevance of data. While the phrase above and other labels like ”the new commodity” or ”the new currency” have been deemed inaccurate it is undeniable that, at this late stage of the Information Age, data has turned to be of supreme importance.

This prevalent role comes from two main factors, the sharp increase in data generation, and the improvement in the quality of insight we obtain from it, for which the cloud and cognitive systems (e.g. Machine Learning) are to thank. In fact, more data has been created during 2017 than in the rest of history combined. While the benefits of it are evident, we should not forget that it is critical for data to be protected. According to the International Data Corporation (IDC), by 2020 67% of the data will require protection, but only 35% will actually be protected. Furthermore, the boost in Cloud usage, both for storage and computing, vastly increases complexity in maintaining security and privacy. The attack surface grows and new risks appear. One of those risks is the loss of privacy on our data, caused by either snoopers or curious administrators. While nowadays is easy to find commercial solutions that tackle the former, the tools to avoid the latter are still rather limited.

More personally, the motivation behind this thesis comes primarily from my research on Data Privacy carried out in Bilkent University on the Spring semester of 2016, and on the Distributed Systems Group of ETH University from August 2016 until January this year. This thesis seeks to elaborate on the efforts made by academia providing a closer look to cryptographic schemes that can enhance data protection. We aim to provide a tool that allows sending and retrieving data through a chrome extension that provides client side encryption while retaining operation capabilities. The extension will allow the user to choose among various modes of operation, depending on the functionality we want the data to preserve. Specifically, it allows AES as well as different schemes that preserve keyword search and addition. An overview of the performance of the scheme will be presented to the user, including an explanation of the pros and cons.

Contents

1 Introduction

As the world transitions into what some already call the *"Data Age"*, its dependence on Cloud Storage and Computing is only increasing. We no longer envision a world where our data is confined to a single device. Life is now essentially mobile, data is expected to be readily available on any device and permanently synced among all of them. This is, precisely, one of the characteristics of the Cloud that has appealed the most to private users. But while its ubiquity might be the principal advantage for this type of users, its scalability is what has allowed it to take the business world by storm.

For Small and medium Businesses (SMBs), it has not only brought a substantial saving in IT equipment, but has also allowed instantaneous infrastructure growth upon demand spikes or rapid business development. Many firms have traditionally seen data centers and IT infrastructure deployment in general as an expensive and complex process that slowed down their operations. Thanks to the cloud that view has shifted and it now facilitates and speeds up virtually every service in the company. This, however, brings up new concerns. It is no longer enough to deal with the security of our devices as our information is out there, delocalized. This heavily increases the attack surface which, in our case, refers to the sum of ways an attacker could retrieve or inject data. Data breaches have put these concerns under the spotlight, as they have turned from anecdotal to commonplace, [1] [2]. Institutions have sped up their efforts to provide new legislation that adapt to the new landscape like the GDPR [3] while cloud users have turned to encryption to recover their sense of control (amid the actions of cybercriminals and state actors).

There are two main items to take into consideration for any rational encryption strategy; how the key management will be done and where the actual encryption process will take place. These decisions are even more relevant when dealing with cloud settings. We have seen how for common SaaS services like Google Drive and Dropbox, home users have opted for tools that provide client-side symmetric encryption on a per file or per folder basis. This approach has the benefit of keeping your file content unreadable for the SaaS provider but it also has several drawbacks. Firstly, it lacks granularity (the file is treated as a single unit, and applying the encryption selectively, for example on a per field basis in a large spreadsheet, is not possible). The second shortcoming is having to make the passphrase, or master key, and the application used to encrypt available on every device trying to access the files in order to be able to decrypt (this diminishes the multi-device approach SaaS thrives upon). Lastly but equally relevant, it completely prevents cloud-side functions like file preview.

Another approach we have also seen, overall in the database world, is server-side data encryption, this is, encryption performed by the cloud provider. While it checks the box for compliance (storing data in plain text is at odds with, for example the requirements for health records storage), the provider holds the keys and so, who and when has access to our data remains out of the user's control . Bring Your Own Key (BYOK) has been claimed to be a solution to this matter, but generating your own key does not suffice [4] when the encryption and decryption is being made by a third party at its will. [5]

Throughout this thesis we will showcase several tools that allow the user to decide to which extent privacy preservation is primed in every case.

1.1 Contributions

The aim of this thesis is offering a condensed view of the existing technologies for privacy preservation in the cloud. For doing so, apart from this document, we have developed a chrome extension that pretends to act as a one-stop-shop for the evaluation of different encryption methods. The extension allows the user to apply an encryption method from among the ones available, which include AES and partial homomorphic algorithms like Paillier. The encrypted data will then be sent to our instance in Amazon Web Services (AWS) and once the transfer is completed the user will be able to see the advantages and disadvantages of each algorithm implementation. This way the user could have a clear image of the capabilities preserved and the cost associated. The chosen environment is the browser, since its the most common tool for interacting with the cloud.

1.2 Outline

In Chapter 2 we provide the necessary background for the complete understanding of this work. In Chapter 3 we review academic work that is intimately related to our thesis. In Chapter 4 we explain the scenario of our work. In Chapter 5 we detail the design and components of our tool. In Chapter 6 we go over the actual implementation of the extension and how to set up our environment.

2 Background

2.1 Cloud

The Cloud is one of the protagonist of the decade. By allowing to obtain the services that come with having an IT infrastructure without having to undertake the purchase and deployment, it has brought an enormous benefit to virtually every business sector.

The main idea behind the cloud is letting the client make use of the equipment of the provider as if it was her own. The client might seek only storage or also computing power. The provider can provision that storage or computing power for the client on demand, bringing scalability to a level previously unknown.

Generally speaking, there are three different ways cloud services can be classified.[1]

The **type of function** it is destined to. It could be only storage, where we can mainly find Database as a Service (DBaaS) and applications like Dropbox. Alternatively the client could be seeking also computing power, either to speed up an application or process you are already running or one exclusively offered by the either the provider or a third party. An example of it are some analytics services, where the storage is a subset of it.

The **type of ownership** of the cloud. Three models have been traditionally distinguished; public, private and hybrid cloud. In public cloud settings the providers were the actual owners of every machine. In a private cloud setting the software was provided and potentially the whole network managed by a third party but the client had to own the hardware. A hybrid setting meant the presence of both models within our organizations. Nowadays however the distinction is not that clear, some public cloud providers offer dedicated hardware being the multi-tenancy no longer a requirement. This and other options have now blurred the lines that separated the different settings.

The **level of transparency to the user/developer**. If what the provider offers us is the provision of the building blocks, like processing, hardware maintenance and networks it will be offering what is called Infrastructure as a Service (IaaS).

[1]Some organizations like NIST devise only two ways, by the type of services or the type of deployment (http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf)

At a higher abstraction level we find Platform as a Service (PaaS) where you are not only provided the hardware but also the software tools (e.g. libraries, frameworks) required to get an app running, either proprietary or purchased. Finally there is Software as a Service (SaaS) where everything is transparent for the user, that simply obtains access to an application that is reality running somewhere else and then accessed, for example, from the user's browser.

2.2 Data Privacy

Even before the advent of computers, certain data was considered sensitive and hence expected to remain private. Some record types like those related to health-care or judicial procedures were already required to be placed under physical safeguards. However, this traditional view of privacy protection started to be insufficient when the information abandoned its physical form with the widespread use of computers, and became almost obsolete with the adoption of the Internet. As the network of networks grew in popularity, the complexity of privacy preservation augmented drastically, requiring even the introduction of new, more specialized terminology like *digital privacy/online privacy/internet privacy* [6].

But, not only controlling the access to the data became more complex, also new kinds of data were generated, like browser usage. New targets like smartphones have also emerged and these new central driver for info exposure, can be often overlooked, failing to realize the amount of services it aggregates. Consequently, users have been forced to analyze further where they put their trust [7] and widen the set of information they consider sensitive, no longer being limited to health or court records. [2]

The usage of the cloud maximizes the complexity of privacy preservation and adds several concerns on its own [8]. While the academia has acknowledged this fact and worked to assess and solve these new problems, [9] users are not fully aware of it. [3] Unlike specific items users mention when asked/surveyed, like not being observed while you surf the Internet and keeping private which webs you visit, it is difficult to exactly pinpoint the threat associated with cloud computing. This is probably to blame for the little understanding of how big the exposure can be.

From what can be observed from papers and surveys on the topic, there seems to be consensus that the challenges in privacy in the cloud steam from the lack of control

[2]In "Privacy in the digital world", based on a survey by Waveston, the answer *Having control over who can get information about you* ranked first when the participants were asked to define privacy in today's world

[3]In the same report, we find explicitly stated that *"technologies which provide citizens with the ability to choose the data they share, such as connected objects or Cloud services storing private information, are considered less risky in terms of privacy"*

a cloud user has [10] [11]. That lack of control can expose the user to a misuse of the data (resale or use for secondary purposes), to unknown replication of the data (for efficient and convenient delivery) [12]. As well as issues derived form those, like unawareness of contractors that might tangle up liability or data residing in several jurisdictions. The most common element to ensure those concerns did not materialize keeps on being the contracts signed with the service provider, usually in the form of Service Level Agreement. However, they are not the only way to tackle this matter, a set of tools and approaches exist [13] [14], aiming at achieving privacy *by default* and *by design* [15] [16].

2.3 Privacy Enhancing Technologies

This set of technologies, often referred to as PETs, is defined as *"...technologies that are enforcing privacy principles in order to protect and enhance the privacy of users of information technology (IT) and/or of individuals about whom personal data are processed (the so-called data subjects)"*

We can find pages listing several of the PETs available, [17] but due to its broad definition, there is no absolute list of the tools it encompasses. In fact, even grouping them in categories can be done under many different criteria. A popular approach is using Solove's privacy taxonomy [18], which outlines the types of privacy problems a user can face. The tools are put together depending on which of those threats they tackle. The categorization can also be done according to the technology the tools make use of. Bobby Vellanki distinguishes the following four categories.

1. **Anonymity tools** Their main aim is to avoid the user to be identified and/or linked to a certain action (e.g. accessing certain file in a database, visiting a specific website, sending a message to a precise recipient). Some of the solutions leveraged by tools under this section are:
    - Proxies: The proxy server relays the requests send by the user, so that the server in the receiving end cannot see where the request was originated. When a Trusted Third Party (TTP) is not an option, the user should perform a minimum set of checks (e.g. whether HTTPS is allowed) to ensure the proxy does not pose a threat.
    - PIR and ORAM: Private Information Retrieval and Oblivious Random Access Machines are two families of protocols that employ diverse methods to access files in a database without this one learning the item accessed or the path followed to reach it.ORAMs also allow writing .
    - Mixing Networks: Hides sender and recipient of a message by aggregating senders and recipients (from 1-to-1 to n-to-m). The messages can be batched and then delivered at specific times to obscure timings.
    - Dummy traffic: Randomly generated packets get mixed with genuine messages to make traffic snooping more difficult .

2. **Filters** The common mission for all these tools is trying to protect the user from unperceived dangers. Some examples of them are firewalls, ad-blockers, cookie-cutters, anti tracking, spam filters and spyware detectors.

3. **Policy tools** The building block for this type of tools are the privacy policy languages. As an example, the Platform for Privacy Preferences (P3P), developed by the World Wide Web Consortium (W3C) external informative, allows the websites to specify their privacy policy using XML. In the same manner it allows users to set their privacy preferences and thus see whether a site aligns with them. This is a so called "external" language, and it is useful for description but only "internal"(i.e. the ones that define the preferences within a organization) languages can be normative and therefore have support for enforcement.

4. **Encryption** Allows data to be transformed into a compendium of seemingly random characters. It is the Swiss knife of privacy preservation, it can help us secure data in its three states. Securing the channel through SSH or TLS gives us security for **data in transit**. Ensuring the data is encrypted when stored (e.g. in a database) provides us with secure **data at rest**. Utilizing the data in its encrypted form instead of its original form brings guarantees to **data in use**. On top of that, it can be also employed for enhancing solutions described above, for example when coupling layered encryption with the concept of mixing networks we obtain **onion routing** which also allows us remain private in the eyes of the proxies.

2.4 Information Security

Often referred to as *InfoSec*, Information Security is responsible for the protection of information and information systems against unauthorized access or modification of information, whether in storage, processing, or transit, and against denial of service to authorized users [19]. These goals are encompassed by the CIA triad, this is, Confidentiality (the information only gets to intended recipients), Integrity (the information remains unaltered) and Availability (the information can be accessed upon request from authorized parties).

The first step towards achieving Information Security is usually performing a risk assessment. The items to analyze at said assessment are the following:

- What threats/adversaries exist?
- What tools/capabilities can they use?
- How attractive is the target? (goal/intent/likelihood)
- What impact would the attack have?

- What safeguards can be deployed to minimize exposure?

In particular, the types of adversaries, their capabilities and the role of safeguards are of special relevance to this thesis.

Honest-But-Curious Adversary [20]. The honest-but-curious (HBC) adversary is a legitimate participant in a communication protocol who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages. This attacker, also called *semihonest* can be formalized [20]. This adversary will be the base for the *curious administrator* we use in our work. However, for us a constructive definition of the adversary's capabilities will suffice. This model also covers the possibility of another tenant gaining access to our resources in a public cloud.

The most common safeguards for preserving confidentiality and integrity are cryptosystems. Encryption schemes aim to achieve *Semantical Security* meaning *Whatever is efficiently computable about the cleartext given the ciphertext, is also efficiently computable without the ciphertext* [21].

For real applications we deal with an equivalent,but easier to prove, version of this notion, the so-called *Ciphertext Indistinguishablility.* It implies that for any two messages ***m0***, ***m1*** the probability of the adversary guessing which message a ciphertext ***c*** encodes is $\leq 0.5 + \varepsilon$, where ε is negligible. This property is typically measured through *games* that attempt to establish how well the scheme copes with specific threats. In those games, the adversary is given certain capabilities, and if the system/protocol is able to retain the condition above, it succeeds. For example, if the adversary can obtain the ciphertext for a certain input known by her, it performs a chosen plaintext attack (CPA). Shall our system succeed under this assumption, it will be deemed IND-CPA, Indistinguishable under Chosen Plaintext Attack.[4] These security notions are the standard way of analyzing Cryptographic Systems.

2.5 Cryptography

Cryptography, from Greek *krypto* (hidden) and *graphos* (something drawn or written) is the science that studies secret communication. Encryption, mentioned in previous paragraphs, is thus to make some information secret or hidden. The encryption process makes use of an algorithm, known as cipher, a message (plaintext) and a key. The output of it known as ciphertext. To decrypt one starts with the ciphertext and using the cipher and key the original message or plaintext is obtained. Depending on whether it takes two different keys to encrypt and decrypt or it can be done with the same key, we will talk about asymmetric or symmetric encryption respectively.

[4]For more on adversaries and notions of security : courses.cs.washington.edu/courses/cse599b/06wi/lecture10.pdf

2.5.1 Symmetric Cryptography

As we have highlighted in the previous paragraph, the main idea behind this type of encryption is that there is only one key, the same key to encrypt and decrypt the data. This symmetric key can be used with block ciphers or stream ciphers.
A stream cipher takes the key and generates a pseudo-random (seemingly random

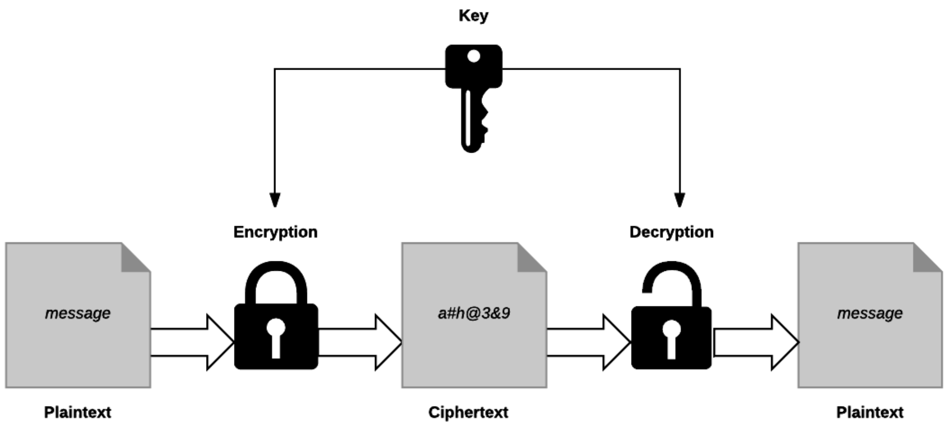


Figure 2.1: Depiction of a symmetric key operation

but not entirely so) keystream from it, that keystream is XORed with the plaintext. This is a bitwise operation, therefore the message, the keystream bits used and the ciphertext will all be of the same length. XORing is a pretty fundamental way of obtaining of a 1:1 map, this bijective relationship is the definition of low *difussion* [22] and makes it suffer from *malleability* (transformation of ciphertext provokes substantial transformation of plaintext). However, it also means that if we were to XOR the ciphertext with the same keystream we would directly obtain the original plaintext. Block ciphers on the other hand operate, as its name indicates, block by block. Apart from satisfying *difussion* principles it also adds *confusion*, which means the relation between the key and the ciphertext has been obscured [22]. The reason why block ciphers present this characteristic/s is tightly knit with the way they function. A block of fixed length from the message is taken along with the key (or a subkey) and subjected to an iteration of rounds in which they undergo a series of processes, they are permuted, swapped, added, etc. The specific processes depend on, for example whether the cipher is based on Feistel Networks * or Substitution Permutation Networks, but the important concept and common characteristic is that we do not need a separate decription algorithm, we can go from plaintext to ciphertext and vice versa with just one. [23]. And this particularity holds true for all symmetric crypto, whether stream or block ciphers.

2.5.2 Asymmetric Cryptography

Symmetric Cryptography has however an inherent problem. When the encryption involves two different parties, sender and receiver, the key has to be present on both ends and it is not easy to find a secure channel for key distribution. In the early 70s, mathematicians from the GCHQ (a branch of British intelligence), started to work on encryption models that were not conditioned by the premise that cryptography needed a shared secret key to be applied at both, encryption and decryption times. [24]. While laying the foundations of it, another advantage could be observed, managing large quantities of key material was cumbersome (overall in the intelligence agencies' sphere), and the proposed model implied the existence of only one secret per user. [25] An actual implementation of the model was first designed in 1973 [26] but the first time concept reached the public community was in 1976 [27]. With implementations of public key cryptosystems published shortly after [28] [29].

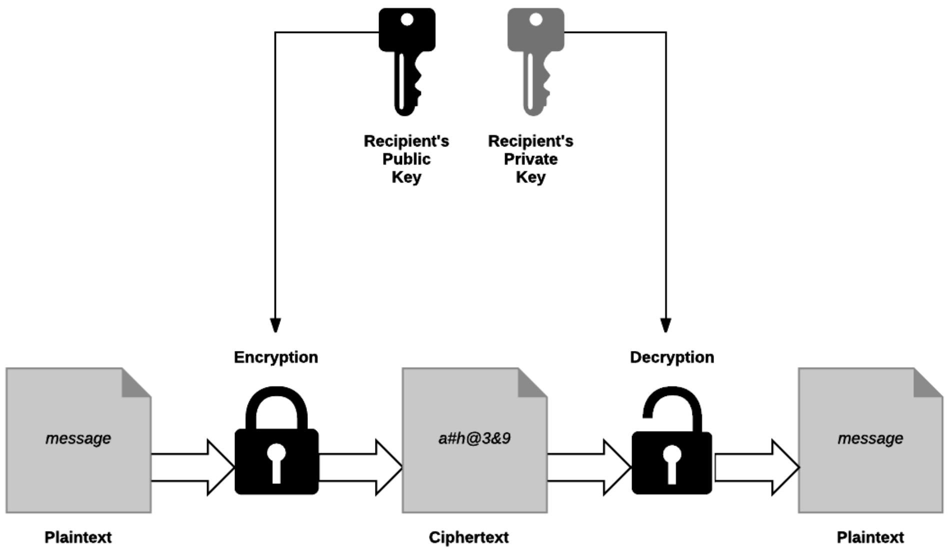


Figure 2.2: Depiction of an asymmetric key operation

In a nutshell, certain mathematical systems/fields like modular arithmetic have some special characteristics that make possible to obtain two keys that despite being different were able to cancel out each other. Yet, the mere fact that this can be done implies that both keys are related, which brings up questions about the feasibility of deriving one key from the other. The algebraic nature of public key cryptograhpy is what provides the assurance that this (the derivation) cannot be done, or at least not in a timely efficient manner. For example, the key A generated through the RSA algorithm cannot be calculated from the key B, since an attacker trying to do so would be facing the factoring problem. This is, it is easy to take two (very large) prime numbers and multiply them, while it is extremely hard to find the prime factors given the big number. These essential, hard to solve mathematical problems are called *hardness assumptions* [30].

2.5.3 Homomorphic Encryption

The possibility of key duality was nonetheless not the only mathematical particularity the RSA algorithm exhibited, a structure-preserving map between two algebraic structures could also be observed. This particularity was acknowledged just a year after RSA was first published and called *privacy homomorphisms* [31]. The homomorphism the RSA algorithm evinces is formally defined as follows:

Let $< \mathbb{Z}_p; \times_p; \equiv_p >$ *be the system of integers modulo p with the functions of multiplication and test for equality. And* $n = p \cdot q$ *, where* **q** *is a large prime and n is difficult to factor. We may take* $\phi^{-1}(x) = x^e (mod\, n)$ *as the encryption function. Since* $(x^e)(y^e) = (xy)^e$ *we have* $\phi^{-1}(x \times y) = (x \times y)^e\, mod\, n = (x)^e mod\, n \times (y)^e\, mod\, n = \phi^{-1}(x) \times \phi^{-1}(y)$

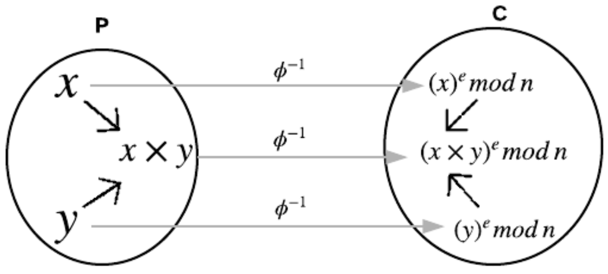


Figure 2.3: Depiction of an homomorphism between two fields

In 2.3 we see a graphic representation of it. Homomorphic encryption can thus be defined as cryptographic constructions that allow to perform certain operations directly on encrypted data, without having to decrypt first and hence keeping the result secret. In particular, the textbook implementation of RSA seen above allows to perform multiplications, we say it exhibits *multiplicative homomorphism*, is *multiplicatively homomorphic* or *homomorphic under multiplication*. Other schemes are *additively homomorphic*, like Paillier [32]. ElGamal [33] has also an additive version despite the original scheme being multiplicative. All the aforementioned cryptosystems have a key characteristic in common, only one type of operation can be performed over the encrypted data. This characteristic makes them fall under the category of ***Partially Homomorphic Encryption (PHE)***. PHE schemes are able to perform an unlimited number of operations with ciphertexts that correspond to an unlimited number of additions or multiplications with plaintexts.

In 2005 Boneh, Goh, and Nissim (BGN) devised a scheme [34] able to perform both, multiplication and addition. The BGN scheme can perform an unlimited number of additions but only one multiplication [35]. These type of schemes provide ***Somewhat Homomorphic Encryption (SWHE)***, defined as systems that can only deal with a limited number of addition and multiplications before decryption fails [36]. This limitation is more formally expressed as a *C-evaluation scheme (Gen,Enc,Eval,Dec) that has correct decryption and correct evaluation* [37], where C is the set of boolean circuits that can be computed. In the case of BGN, this allows performing functions of polynomial order 2.

Seeking the scheme that allowed unbounded additions and multiplications became of supreme importance, since it could perform any kind of operations. This is called ***Fully Homomorphic Encryption (FHE)*** and the first plausible implementation of it came in 2009 by Craig Gentry [38]. His approach was starting with a SWHE scheme that added noise on every operation and develop a method that allowed reducing that noise before the ciphertext became irrecoverable (bootstraping). This was not only a breakthrough that encouraged further development of FHE systems [39] [40] [41] but also inspired methods like the developed by Catalano et al. [42] that applied to the typical PHE increases the number of operations that can be performed. However, despite the constant refinement of FHE schemes, it is still too slow for practical implementations.

2.5.4 Functional Encryption

One of the most important premises of Homomorphic Encryption is that the party performing the computation does not learn the output of it, since it remains ecnrypted. However sometimes it might be desirable for the server to be able to read certain outputs (e.g. spam filter over encrypted e-mail). Furthermore, an operation's performance could be negatively affected by the *FHE* constrains, for example, binary search benefits from knowing the data structures.

Functional Encryption (FE) is the name given to the family of cryptosystems that allow a receiver to obtain the result of applying a function on an encrypted input, without revealing any other information about the input. This type of encryption was first mentioned in 2008 [43] by Waters and formally defined in 2011 by Boneh, Sahai and Waters [44]. Having its origin on the concept of Public Key Cryptography, FE is actually a generalization of Identity-Based Encryption (IBE) [45], Attribute-Based Encryption(ABE) [46] and Predicate Encryption (PE) [47]. FE relies of four algorithms, *Setup, KeyGen, Enc* and *Dec.* In general, three parties are present in the process; the owner of the message *(Alice)*, the receiver that wants to learn the outcome of a certain function *(Bob)* and *Master Authority* that grants control over what functions can be applied to the data, issuing keys for appropriated functions. Figure 2.4 provides an overview of how *FE* operates.

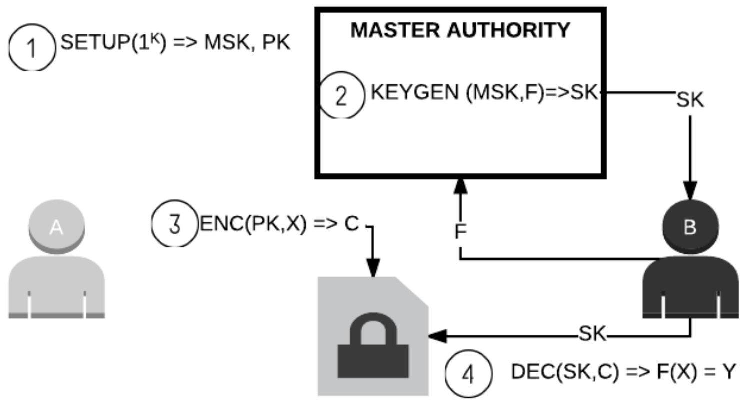


Figure 2.4: Depiction of the parties and algorithms involved

There are several possible constructions of *FE* schemes. The main variations among them are the underlying assumption they use (e.g. multi-linear maps) and the function to be computed (e.g. inner products). Later in the thesis we will look at systems that implement Symmetric Searchable Encryption (SSE) which is a particular *PE*.

2.5.5 Property-Preserving Encryption

Applying encryption to a plaintext can result in a ciphertext that does not resemble the original message at all, not only in terms intelligibility, but also in its format. While the former can be solved by removing the randomness, the alphabets might still be different(numbers do not necessary map to numbers) and this can have a significant negative impact on Database systems. This problem was first addressed by Brightwell et al. [48] who aimed at maintaining the datatype despite encrypting. The foundations of ***Format-Preserving Encryption (FPE)*** were laid by Black and Rogaway in 2002 [49]. It is worth mentioning that block ciphers exhibit indeed this preservation, they map $\mathcal{K} \times \mathcal{X} \to \mathcal{X}$. This however is limited to binary domains of the form $\mathcal{X} = \{0,1\}^n$ being n one of the predefined sizes available for the block cipher. The goal is then preserving the format for an arbitrary domain $\mathcal{X}$.
There are various FPE constructions for $\mathcal{X} = \mathbb{Z}_N$. It is possible to use a block cipher whose block size is close to the size of the domain we want to encrypt and then do *cycle-walking*.[5] Another method is using a Feistel network, the message $m \in \mathcal{M}$ is decomposed into two numbers a, b and used as inputs of the network. As we can observe in the part *a)* of figure 2.5, the classical Feistel has binary strings as input and output, so *cycle-walking* might still be needed. To circumvent the inconvenience,

[5]Let $M = [0, 10^6]$. Then $n = 20$. Suppose we have a 20-bit block cipher on the set $\mathcal{T} = [0, 2^{20} - 1]$ We wish to encipher the point $m = 314159$; we compute $c1 = EK(314159)$ which yields some number in $\mathcal{T}$, say 1040401. Since $c1 \notin \mathcal{M}$, we iterate by computing $c2 = EK(1040401)$ which is, say, 1729. Since $c2 \in \mathcal{M}$,we output 1729 as $Cy_K(314159)$. For efficiency we seek $|\mathcal{T}| \simeq |\mathcal{M}|$

one can directly partition the decimal number and use it as an input as in *b)* .

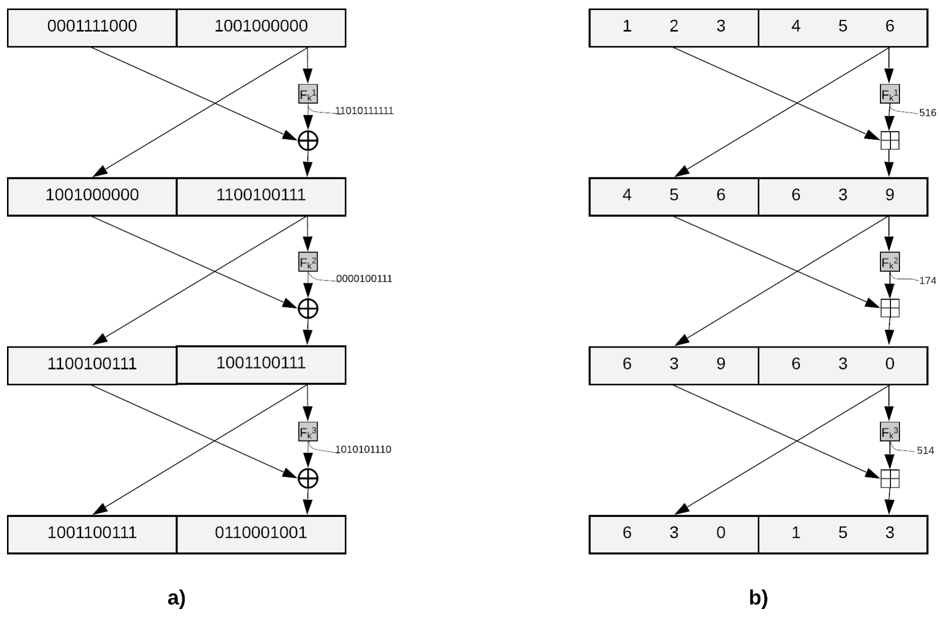
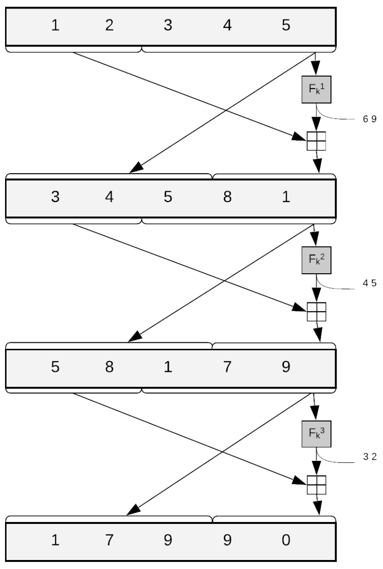


Figure 2.5: Feistel Networks for even inputs

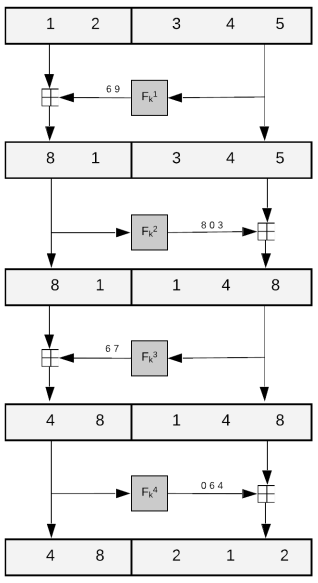
Constructions *a)* and *b)* only work for even inputs. For odd inputs, we count with two alternatives, unbalanced networks and alternating networks. The former "borrows" digits from left to right or viceversa depending on the round number, the latter applies expanding or contracting functions alternatively, also depending on the round number. They are c) and d) of the figure 2.6 respectively. The aforementioned functions are usually some flavour of AES or similar, therefore they will output bit strings. The solution adopted for obtaining a string of decimals instead is to take the last d bits of the output modulo 10^m, the number d will depend on how many decimals m our construction requires (e.g. for $2 \leq m \leq 9$ we will take $d = 64$ bits). We must also note that the XOR present on classical Feistel does not make sense outside a binary domain, instead we must use some other operator. For decimal alphabet we might use characterwise addition. This means that, character by character, we add up, modulo 10, the corresponding characters of the equal-length strings L and $F_K^1(R)$ (carries are ignored). The second possibility is blockwise addition.

These schemes satisfy the requirements for decimals, albeit not random domains. In 2009, Bellare et. al [50], revisited this concept and apart from providing FPE with a formal definition, they made two crucial additions. First, they introduced *tweaks* (distinct, publicly known inputs) in the constructions. [6] And second, they suggested

[6]The importance of tweaks can be observed when applying FPE to credit card numbers(CCNs). For CCNs, the first six digits and the last four need to be available to the application, so only six digits are encrypted. Given that FPE schemes are deterministic, if we were not to add the



c)



d)

Figure 2.6: Feistel Networks for odd inputs

a method for using the Feistel-based schemes as building blocks for achieving *FPE* on any arbitraty domain $\mathcal{F}$. They named the approach *Rank-then-Encipher (RtE)* and as its name suggests it is based in mapping the elements of $\mathcal{F}$ into $[N]$ (a form of indexing) so that *FPE* on integers can be applied. Although highly relevant, the *RtE* method only gave an abstract idea on an actual implementation and it was proven inefficient and insecure for Weiss et al. [51]. As of today, two designs based on *RtE* are truly generic and thought to be secure, *LibFTE* [52] and *GFPE* [51]. Both of them use the NIST-recommended methods [53] for *int-FPE*.

Order-Preserving Encryption (OPE) is another property-preserving scheme often sought after for its convenience. Under *OPE*, the order of the plaintexts is preserved after encryption. Formally, for $m_1 < m2$ we have $c_1 < c_2$ where $c_1 = Enc(m_1)$ and $c_2 = Enc(m_2)$. A trivial approach one might take towards OPE is simply selecting a constant K and adding it to every plaintext such that $c_i = p_i + K$. While this does indeed preserve order it is obviously inherently insecure. Bebek et al. [54] gave this idea a twist, suggesting a system where $c_i = \sum_{j=0}^{p_i} R_j$ where R_j is the output of a random function. This system presents important drawbacks, it is computationally expensive and reveals the distribution of the plaintexts. Gultekin et al. [55] based their scheme in a polynomial whose coefficients were derived from a secret key but revealing the distributions was still a problem. As Aggraw et al. mention in their paper [56], the intuition one might have to tackle this recurrent

tweak we would only have 10^6 possible encryptions. This means that in a large database we would have different CCNs having the same ciphertext, something to be avoided.

distribution problem could go along these lines: Generating $|P|$ unique values from a user-specified target distribution and sorting them into a table T. The encrypted value c_i of p_i is then given by $c_i = T[i]$.The decryption of c_i requires a lookup into a reverse map. Here T is the encryption key that must be kept secret.This simple scheme, while instructive, has the following shortcomings for it to be used for encrypting large databases: The size of encryption key is twice as large as the number of unique values in the database. Updates are problematic. When adding a new value p, where $p_i < p < p_i + 1$, we will need to re-encrypt all $p_j, j > i$. The approach in [56] is more complex, it requires first analyzing the input distribution, then dividing in in sections (buckets)and using a mapping function to make it *flat* (uniformly distributed). They also select a target distribution and do the process backwards. Bucketize the target distribution, flatten it into a uniform distribution and then reconcile the sections of both flattened distributions, scaling them in such a way that their widths become equal. Their system is designed for a setting where users know all data in advance. Hence, the encryption algorithm must take all the plaintexts as input.

The first provable security analysis for *OPE* was performed by Boldryeva et al. [57] where they also present an alternative scheme. Their scheme is based in the Hypergeometric (HG) distribution. In the papers is noted that any order-preserving function *(OPF)* f from $\{1, ..., M\}$ to $\{1, ..., N\}$ can be uniquely represented by a set $\mathcal{S}$, a combination of M out of N ordered items. Hence, by mapping each $i \in [M]$ to the ith smallest element in $\mathcal{S}$, an *OPF* is constructed. For their construction, they use what is called "lazy sampling". Given $i \in [M]$ the algorithm starts by dividing the target space (range $\mathcal{R}$) by half and calling this point y. Then the HG distribution outputs the *domain gap*, an $x \leq y$ that describes the number of points of *OPF* less than y. Since the ith element of the *OPF* is the ciphertext of i, if $i < x$ we need to recur on the points of the domainless than or equal to y. If $x < i$, we recur on the points of the domain greater than y. When our domain only has one point remaining, the range will be a set of points in which our ciphertext can reside, the algorithm will simply select one and output it.
There exist a number of *OPE* protocols that are interactive and rely on common indexing structures. Specific mention to them is ommited from this chapter as they will be analyzed in more depth in coming chapters.

3 Related Work

In this chapter, we give a closer look to work on academia that is specially relevant to this thesis, namely at the field of computing over encrypted data. In particular, we analyze two lines of work. First, systems that allow a considerable number of operations over encrypted databases, and we take CryptDB as the most representative. And second at the efforts of researchers for providing a somewhat standardized way of characterizing and analyzing such systems.

3.1 Searchable Encrypted Database Systems

Propelled by the database (DB) community, searching over encrypted data has became a recurrent research topic for many universities. This subject, which builds upon some of the crypto primitives we have outlined in the background section, has yielded efficient and optimized indexing and flexible query support (e.g. for numerical range, comparison, or aggregation queries) which in turn have allowed the emergence of encrypted DB constructions that preserve all that functionality.

3.1.1 CryptDB

One of the best examples of such systems is the construction form Popa et al [58]. CryptDB has been wildly successful because it can execute a wide range of SQL queries over encrypted data. It leverages the fact that most SQL queries use a small set of operators, supported over encrypted data in their system.

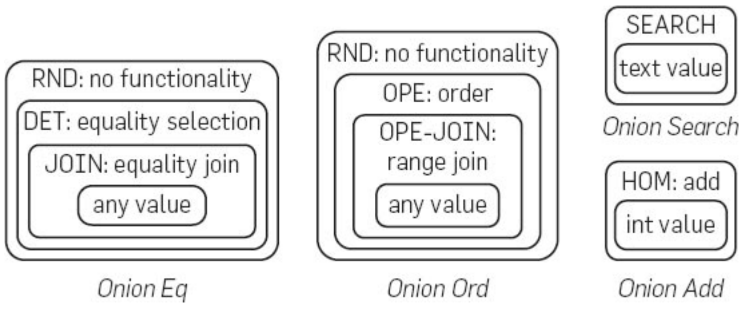


Figure 3.1: Encryption layers in CryptDB [58]

Two concepts are key to understand CryptDB, the layered encryption (in an onion fashion, see figure 3.1) and the role of the proxy that handles the query re-writing process. Using layers of encryption ensures that, while the number of schemes present allows the system to perform a large amount of operations, the exposure only depends on the amount of functionality required by the queries.

It uses AES or Blowfish (for integers) block ciphers in CBC mode for the *Random (RND)* layer. The same block ciphers but in CMC mode for the Deterministic (DET) layer. For the *Order preserving (OPE)* layer they use the scheme by Boldryeva et. al in their first implementation and the mOPE [59] they developed is their releases after that. The Paillier algorithm is used for the *Homomorphic (HOM)* layer. A scheme of their invention is used for the JOIN layer and the method by Song. et al for the *SEARCH* layer. Having all this schemes available means that the DBMS (Database Management System) can answer queries to the encrypted database like the ones making use of: GROUP BY, COUNT, DISTINCT, (=)?, GROUP BY, COUNT, DISTINCT, RANGEQUERIES, SUM, (+),LIKE (wordseach).

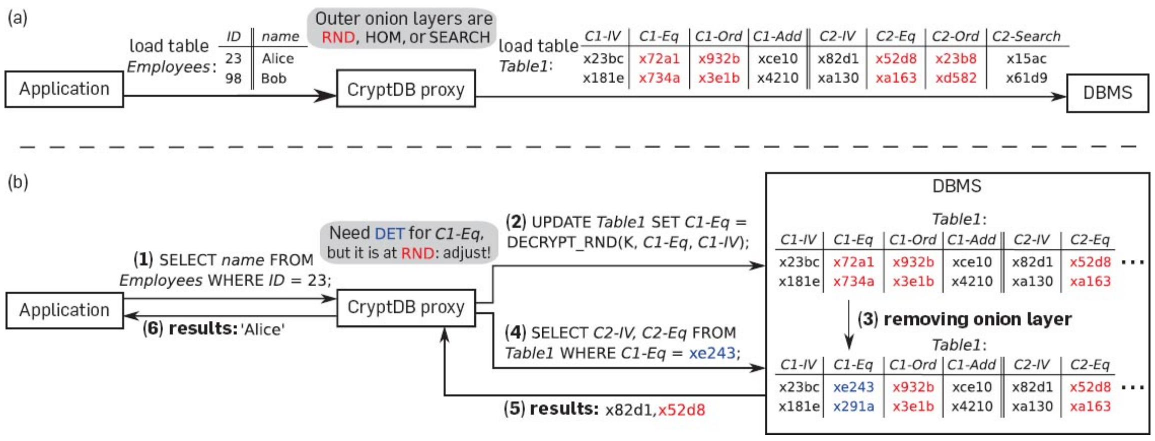

(a)
load table Employees:
ID | name
23 | Alice
98 | Bob
Outer onion layers are RND, HOM, or SEARCH
load table Table1:
C1-IV | C1-Eq | C1-Ord | C1-Add | C2-IV | C2-Eq | C2-Ord | C2-Search
x23bc | x72a1 | x932b | xce10 | x82d1 | x52d8 | x23b8 | x15ac
x181e | x734a | x3e1b | x4210 | xa130 | xa163 | xd582 | x61d9
Application
CryptDB proxy
DBMS
(b)
(1) SELECT name FROM Employees WHERE ID = 23;
Need DET for C1-Eq, but it is at RND: adjust!
(2) UPDATE Table1 SET C1-Eq = DECRYPT_RND(K, C1-Eq, C1-IV);
(4) SELECT C2-IV, C2-Eq FROM Table1 WHERE C1-Eq = xe243;
(5) results: x82d1,x52d8
(6) results: 'Alice'
Application
CryptDB proxy
DBMS
Table1:
C1-IV | C1-Eq | C1-Ord | C1-Add | C2-IV | C2-Eq …
x23bc | x72a1 | x932b | xce10 | x82d1 | x52d8
x181e | x734a | x3e1b | x4210 | xa130 | xa163
(3) removing onion layer
Table1:
C1-IV | C1-Eq | C1-Ord | C1-Add | C2-IV | C2-Eq …
x23bc | xe243 | x932b | xce10 | x82d1 | x52d8
x181e | x291a | x3e1b | x4210 | xa130 | xa163

Figure 3.2: a)CryptDB's transformation and encryption of the database b) query flow showing onion adjustments. Ciphertexts denoted by "x. . . " [58]

At the very beggining the proxy encrypts the database with the "onions" (one or more per data item) leaving them in their strongest encryption layer (see figure 3.1 for reference). Upon receiving SQL queries from the application, the proxy intercepts it and rewrites it. It anonymizes table and column names, and encrypts the constants in the query. Some operations are replaced with User-defined functions (UDFs). Then, it evaluates in which layer the specific column or columns targeted by the query is. If the current layer of the column is not already at an onion layer that allows the predicate, the proxy sends an UPDATE query at the DBMS server, which invokes a UDF to "peel" it to the necessary layer. Once the layer is adjusted the encrypted query is sent to the server and executed. Lastly, the proxy will decrypt the result of the query and return it to the application. This system flow can be observed in figure 3.2.

3.2 Structured system evaluation and comparison

Evaluation is a part of high importance for publications in computer science. Looking at the security your system provides and the performance it has is even more vital in the field of the computation over encrypted data, given the difficulty of balancing security, performance, functionality and usability . While evaluation is present in almost every publication, the comparison tends to be done with the works in their very specific subset(like the comparative tables in some papers on SSE [60] [61] or OPE [59]).This tendency is in great measure caused by the difficulty of comparing works which might not have the same background or target but unfortunately this comparison are not enough to properly characterize a system like the one we have described in the previous section and also makes hard to get the bigger picture.

3.2.1 SoK

The so-called SoK (Systematization Of Knowledge) is a line of research originated at the IEEE Symposium on Security and Privacy ("Oakland" conference) in 2010. It is indicated for fields whose body of research has grown to a size where a systematization of it is necessary for a better general understanding. In particular, the recent SoK paper [62] on cryptographically protected database search is specially useful for the purposes of our thesis, since it provides clear analysis parameters. It characterizes protected database systems (umbrella term for both SSE and PPE) and their underlying cryptographic tools through a structured, well-defined analysis[1].

We go through some of the components of such analysis. Like the strictly defined leakage profiles, which define the following objects within a protected search system as vulnerable to leakage:

- Data items, and any indexing data structures.
- Queries.
- Records returned in response to queries, or other relationships between the data items and the queries.
- Access control rules and the results of their application.

And the information that can be leaked from those objects listed in increasing damage:

1. Structure: properties of an object only concealable via padding, such as the length of a string, the cardinality of a set, or the circuit or tree representation of an object.

[1]They restrict the analysis to protected database search systems that provide formally defined security guarantees based upon the strength of their underlying cryptographic primitives

2. Identifiers: pointers to objects so that their past/future accesses are identifiable.
3. Predicates: identifiers plus additional information on objects. Examples include “matches the intersection of 2 clauses within a query” and “within a common (known)range.”
4. Equalities: which objects have the same value
5. Order (or more): numerical or lexicographic ordering of objects, or perhaps even partial plaintext data

They provide a classification of the cryptographic schemes said systems rely on, distinguishing as categories and subcategories:

1. The Legacy Index : Can be used with plain normal DB, it just modifies the data inserted and the queries.
2. Custom Index : special-purpose protected indices and customized protocols that enable the querier and server to traverse the indices together (less leakage)
    - Custom Inverted Index
    - Custom Tree Traversal Index
    - Other Custom Indices
3. Oblivious Index : Custom index but also hides repeted retrieval of same record/object. This schemes typically hide data identifiers across queries by re-encrypting and moving data around in a data structure.

Performance and usability are described along three dimensions:

- The scale of updates and queries that each scheme has been shown to support.
- The type and amount of cryptography required to support updates and queries.
- the network latency and bandwidth characteristics.

We also find useful the mention of the benefit every scheme provides. Expressed though query primitive they allow (Equality, Range, etc).

4 Objectives

This chapter further develops the scenario considered in this thesis and the threat model for it. We also explain how our work seeks improving the privacy and security of cloud users, and we analyze our approach in the context of the related work we have presented in the previous chapter.

4.1 Scenario

We contemplate a very common setting for cloud services users as our scenario, the two-party setting. In this setting the *User* will be the one uploading the data to the (Cloud) *Provider* and that same *User* will, at a later time, send queries to the *Provider* and get some data back. This is a fairly simple model that can be schematized like at the figure figure 4.1.

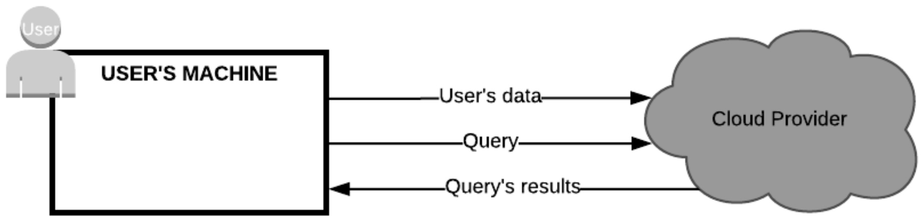


Figure 4.1: Generalization of our scenario

4.1.1 Threat Model

Looking at the diagram above, we can observe how it is necessary to worry about the data in its three states:

- Data in transit/motion [1]: If the communication *User-Provider* is done over an insecure channel, an attacker could gain access to our data by just sniffing the (data) traffic. This is easily solvable by securing the channel through SSH or TLS (HTTPS).

[1]While commonly understood as the same thing, there is a growing trend in the security sector to reserve the term *data in motion* to the data in preparation leading up to transmission and breakdown after transmission [63]. It is more accurate then, to say data in motion only when, for example, the data is being read from the disk by the application.

- Data at rest: If once the data has arrived to the provider it is stored in its plaintext form, any access (potentially a breach) to the *Provider's* database would expose our data. This is solved by storing the data encrypted.
- Data in use: Even if the two previous states of the data are protected, when the *Provider's* DBMS needs to perform any operation (e.g. adding the value of two records) it will typically* perform it over the unencrypted data. Any vulnerability on the *Provider's* system will mean our data can be at risk.

When considering an external attacker, it seems like just the data in use is at risk. Secure channels have been thoroughly studied and efficiently implemented so as long as we use the solutions described we are not to worry. In the same manner, the *Provider* encrypting and decrypting records in the database is an easy and efficient process. However, since we will be working with the HBC adversary model, the *Provider* is not trusted and therefore both the second and third states must be addressed. In order to keep the confidenciality of our data in this scenario we must ensure that the *Provider* only sees our data in its encrypted form at all moments.

4.2 Analysis of related work

In the previous chapter we have seen highly relevant work for implementing and methodically analyzing protected database systems. In this section we will detail why their work could not be directly used to meet our goal. In our use case, a person interested in this field (who is not necessarily a researcher) wants to deepen her knowledge on protected search. [2]

4.2.1 CryptDB

There are two main reasons why CryptDB is not well suited for our use case. First, while CryptDB is not specially hard to deploy (in Ubuntu), it is aimed at providing functionality and not information about its inner components. And second, even after considering the additional information that can be obtained from their paper (which we summarize in Chapter 3), the reader would only obtain a general idea of a scheme of each kind, you could not compare different approaches to some of the primitives.(Not able to put them in context).

4.2.2 SoK

Despite providing a great set of definitions that allow to analyze the schemes in this field in a more objective and structured way, the major part of tools characterized in their paper was assessed by analyzing the data given by they authors themselves.

[2] All the systems really revolve around the same key elements, a type of encryption that allows you do keyword search, one that lets you sort,etc so by getting to know about these core elements one would acquire a good understanding of the field.

A subset of them was actually implemented and tested through a web interface in a pilot study. That testing interface is no longer available. We also need to mention that the purpose behind their testing was determining whether those schemes were already fit for industrial application. We intend to be provide a much simpler tool (as few intertwined files/dependencies as possible), that ideally allows the user "plugging" some other future, novel schemes and have them alongside the ones we analyze now for an easier side-to-side comparison.

5 Design and Implementation

In this chapter, we elaborate on our tool's design, address its components and how to implement them. First we will state our goal and provide a more detailed diagram of our system/scenario. Then we will analyze the elements on the user's site, the ones on the left-hand side of the diagram. We will provide a reasoned discussion behind the platform choice and the particularities of such platform. After that, we will give an overview of the cloud environment where our database will reside and we will end the chapter explaining the extension's usage model we had in mind during our design process.

5.1 Design goals and system overview

As we have mentioned in the previous chapter, the purpose of our tool is not providing an analysis as exhaustive and complete as the one mentioned in the SoK paper [62]. We rather seek providing the users a better understanding of the building blocks of the protected search technology, so they can better approach and assess both, systems from the academia and commercial systems.

To do so, our work must meet the following goals:

Portability and ease of deployment Getting our system up and running should require minimal effort. The deployment of the back-end should reside in a Cloud Provider (to better showcase the intended usage of the technologies) and therefore some configuration will be needed. For that reason the part of the tool the user faces should be as compact as possible and operate transparently.

Modularity The tool should be as little convoluted as possible, reducing this way the effort a user would have to do in order to plug-in and test other schemes (either newly published or not implemented by us).

Ease of comparison The different types of schemes should be analyzed according to the same parameters so that the user can easily compare two different schemes even if these belong to different categories.

Impact The web environment has experienced a rapid development. The browser is now the most important player in everything internet sphere and the emergence of companies like sync and pCloud who provide encrypted cloud and do actually all the process on the browser proves it. Although not a goal per se, we like to think it is possible that our tool helps making this special (searchable) crypto more approachable and this can lead to it being rolled out on the browser. Ideally helping them to come up with more complete/more efficient/ more secure products.

To talk about our design more in detail we need also a more detailed diagram. The figure 5.1 found below, shows the same scenario as in the previous chapter but including the components of our system[1].

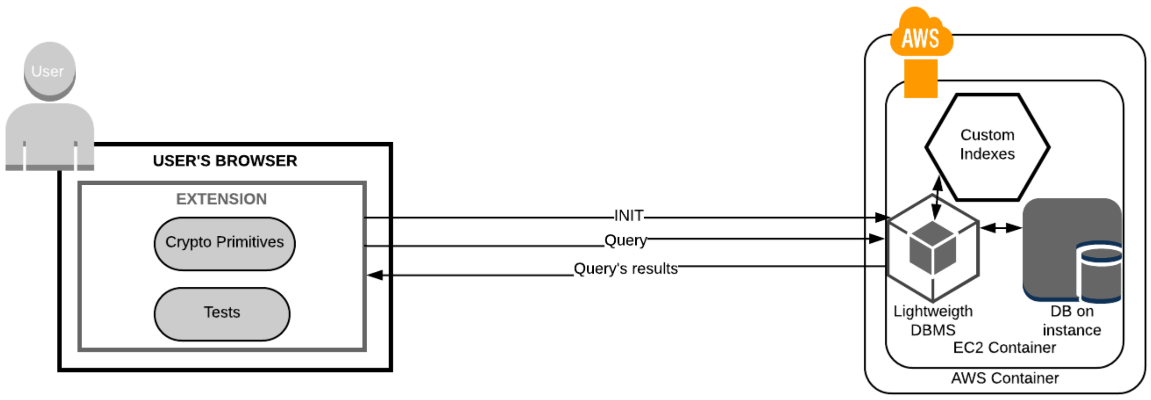


Figure 5.1: A more detailed depiction of our scenario

5.2 User's Browser

Using a browser is the way most people access the web and that makes them an extremely popular piece of software. That popularity makes any tool based on them have a great availability which effectively benefit its impact. On top of that, browsers could to some extent be considered OS independent. This fact completely aligns with our goal of portability and ease of deployment.

While some years ago would sound impossible to run any mildly resource intensive application on a browser, they have become more and more powerful and it is currently feasible. The negative side of this transformation is that the increase in complexity has exposed browsers to many security threats. In response, it has been tried to preserve web security by tightening the rules on matters like *same origin policies* or *cache control* for the websites.

[1]Common, necessary elements of the extension like the UI and server communication are part of our tool but not the focus of this thesis and therefore not depicted

5.3 Chrome extension

Those restrictions can affect websites (and therefore WebApps), whereas extensions have remained pretty unconstrained. Other benefits from using a extension are the access to special resources within the browser(chrome APIs) and minimizing the infrastructure needed. When not distributed directly through Google's platform, extensions can bundle together all their elements and that bundle (i.e. unpacked extension) can be directly added by the user to her browser. Integrating thus the tool within a program she is already familiar with and hence facilitating its usage. The reason why we have chosen Chrome extensions in particular is mainly its popularity. Chrome is the most used browser with a 55% market share cross-platform and a 65% share for Desktops. We should also point out that it is feasible porting a chrome extension to other browsers like Firefox [2]. The extension contains both, the User Interface (UI) and the logic of the application.

5.3.1 JavaScript and Cryptography

Javascript crypto has been deemed insecure and even harmful [64], with some experts emphasizing its vulnerabilities and how they could potentially lead to attacks [65]. This has caused certain a reluctance to doing cryptography in Javascript (JS), and therefore on the browser/node environment. This rejection started to decay with the integration in browsers of WebCrypto API among other advances, and since, we have seen interesting developments in this area like the ones mentioned at the beginning of this chapter (Sync and pCloud). Despite the fact that those concerns are in general no longer valid [66], they are specially not applicable to our work. This is mainly due to us trusting the crypto libraries we implement, since our purpose is specifically testing them.

5.3.2 Interfaces, Bindings and Transpilers

There are numerous programming languages available for developers and each of them have its strengths and weaknesses. However, at the browser we are pretty much obliged to using Javascript, which, due to its untyped nature [**?**] is not particularly suited from heavy computational tasks. Unfortunately, cryptography is notorious for being computationally intensive which makes JS implementations of some algorithms very slow. The WebCrypto API we have mentioned few lines above vastly enhances the performance for common crypto algorithms [67], this is done by implementing this functions directly within the browser logic and exposing the high level functions to the JS code. Developers are able to obtain this *interface* thanks to a *binding* [68] between the inner language of the browser (C) and Javascript. But not many functions requiring heavy computation have been added to the browser engine, and even the algorithms provided by the aforementioned API have taken a

[2]https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Porting˙a˙Google˙Chrome˙extension

long time to be widely offered by the major browsers.

Although a developer can choose to include a library in a different language and create a *wrapper* of her own that binds it with JS it is certainly not a trivial task. Avoiding having to do so is one of the main motivations behind *transpilers*, a transpiler takes the desired library as an input and outputs its equivalent code on the target language. In our work we will be using *transpilers* not just for performance reasons but because many of the algorithms we want to be tested in the future are novel approaches, just implemented in one language (which often is not JS).

5.3.3 Implementing the Chrome Extension

As mentioned couple paragraphs above, an extension requires of a UI. Just like in a webpage it is composed of *html files* that provide the content to be shown, *css files* that add styling to the content and *javascript files* that allow the content and styling to become dynamic. To increase the usability and the ease of development we use a *UI Theme* that gives us ready-to-use styling and dynamism. This way we have mainly had to create the html files for our extension. These html files are depicted in an appendix at the end of this document.

Despite the importance of the UI for usability, the actual purpose of this project is the functionality provided by the logic of the extension. Everything happening on the extension is orchestrated in the *script.js* javascript file. The dependencies *script.js* requires in order to operate successfully are depicted in a diagram in the appendix. Most of the functioning of the file is also explained at the appendix, however, the explanation of how *script.js* communicates with the Server is omitted since we have made use of the *AJAX* method of the well-known library *jQuery.* The AJAX (Asynchronous Javascript XML) method is just a more completet yet simpler to use version of html's *XMLHttpRequest*, with the advantage of being handled asynchronously.

5.4 Amazon Web Services (AWS)

We deploy our back-end in an AWS instance. We have chosen working in an IaaS environment since, as explained in the background section, it provides us with the necessary resources and a great freedom of implementation.

5.4.1 EC2 Instance

We have chosen our instance (virtual machine) to run on Amazon's version of Linux, *Amazon Linux AMI.* Since we access the instance from Windows, we need to use a program to establish the connection to our instance. We have chosen to use *Putty* as it allows us to connect using SSH (Secure Shell) in an easy and effective way with the keypair generated during the set-up process. In this instance we deploy the server-side part of our application (implemented in Node).

5.4.2 Configuring our instance

Once we have created our AWS account and gone through the instance type selection wizard, our instance gets prepared and goes online automatically . By default, only the SSH port is open. We change that in the tab *Security Groups* inside the *Network and Security* options. We open the port 8081 because we have programmed our Node application to listen to it for the file transfer and the port 80 to allow serving HTTP. The IP address allocation for our instance is dynamic, which means that if we were to stop the instance and start irt again, we would potentially have a different IP address. This can be avoided by assigning a so-called *Elastic IP* which is simply an static IP address pointing to the instance. We do not require it since we do not stop our instance.

5.4.3 Node backend

For continuity and also due to being already familiar with Javascript, we choose Node for our back-end. Since this part of our application is meant to emulate the Cloud Provider in our scenario, we keep the logic very simple, handling almost exclusively the connection and data transfer with the client and the data storage and retrieval on its local relational database.

For that emulation to take place, we need to make our node app reside in our AWS instance. Since Windows does not natively support the secure copy (SCP) protocol for file transfer, again we are in the need for a dedicated program to transfer it and we again resort to the solution by *Putty*, *PSCP*.

5.4.4 Additional structures

Although ideally the extension's client code would be self-contained, some of the experimental systems require from extra parts like an oracle or additional structures (e.g. B-trees). For implementing those systems (see Future Work) we should allocate those in the EC2 as well, opening this way the system also to schemes that require them.

5.5 Usage Model / Intended operation flow

The standard user usage we have considered when designing our extension can be summarized through the following points:

1. The user goes to **Settings** and adds the IP of the back-end server

2. The user goes to **Encrypt** and select one of the schemes. Each scheme has some files previously prepared for it, the user selects one of those and clicks Encrypt and Send. (The Extension's logic detects which scheme has been requested and reacts accordingly, it prompts an error if the wrong file has been selected and creates the necessary additional structure if needed, after that it encrypts the file/data according to that scheme and sends it to the server where it is stored in the DB). A "successful" message is displayed if everything went ok.

3. The user goes to **Evaluate**, there she sees a list of the files successfully sent and the characteristics of it (size, time and scheme used) as well as an *analysis* link. When clicking that link stats, assurances and graphs showing the performance are displayed.

We make our model more explicit through the *use case* [69] in the figure below[3], these use cases are a comon tool in software development and normally modelled using UML (Unified Modelling Language) [70].

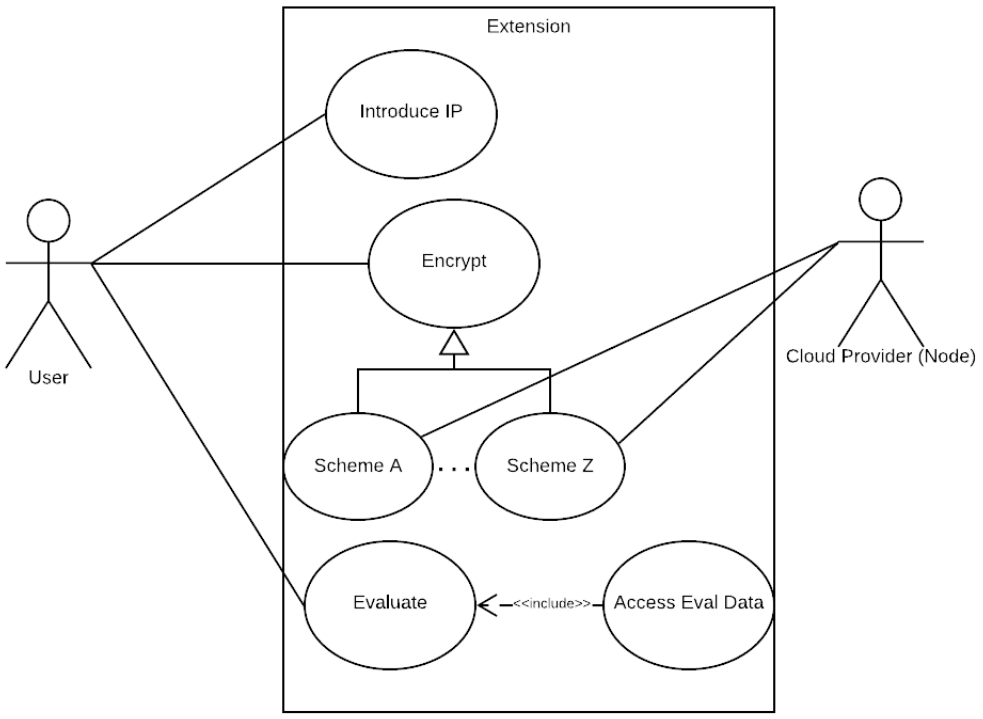


Figure 5.2: UML model of our use case.

[3]This UML model is just from a user's prespective and hence some processes that are transparent to the user do not appear here.

6 Conclusion and future work

6.1 Conclusion

Javascript environments are no longer hostile to cryptography , the paradigm has shifted and we even see some companies that base their client-side crypto in frameworks like angular. With our project we have shown that it could be taken a step further and benefit from the added functionality of special cryptography like partially homomorphic, while still basing the client entirely on the browser and without trusted third parties required. Further developments by the industry could result in services that respect the privacy of the user without the functionality handicap and yet, are very accessible. The fact that our project shows the contrasts between different types of schemes also helps creating a more crypto-savvy community, which in turn increases the awareness of the risks of traditional cloud storage services.

6.2 Future work

We distinguish two lines of work that could help spur the benefits we have acknowledged in our conclusion:

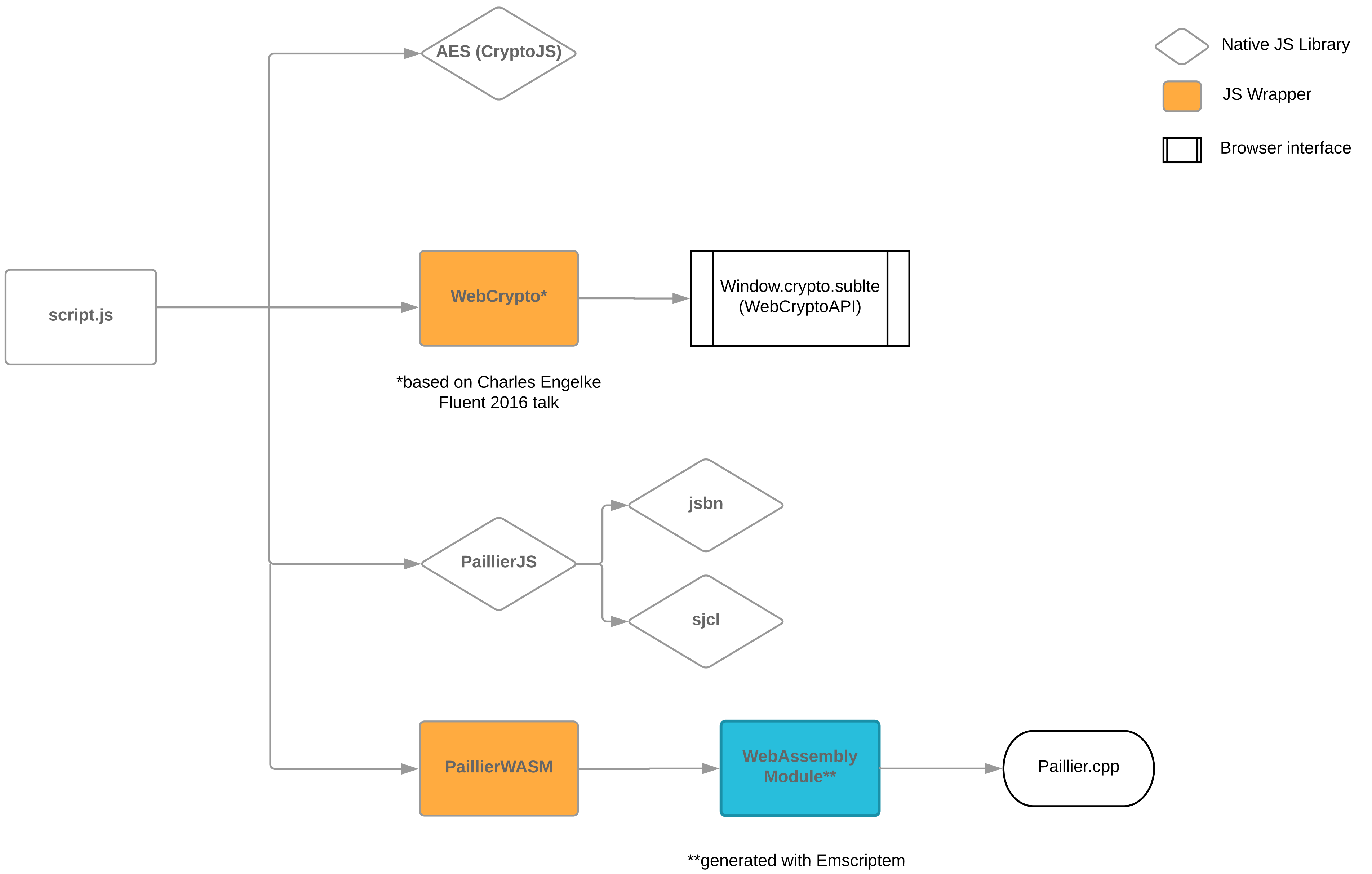
1. Evaluation : If we wished to explore more in depth the educational aspect of the encryption, we should focus on providing a more comprehensive analysis of the schemes. A very promising path to achieving this could be the integration of parts of the SPARTA [71] suite, that has already being used to evaluate research prototypes for secure and private information retrieval systems.

2. New schemes : For future development more geared towards maximizing the impact of our tool in the creation of more complex commercial solutions the focus should be adding other schemes. Specially new schemes that offer more advanced functionality, this would help them make a faster transition from the academia to the companies and hence the general public. A good example of an addition of recent schemes could be some Order Preserving schemes like POPE [72].

Appendix (in the coming pages)

NOTE: Libraries necessary for basic functionality like
jQuery are not explicitly mentioned

Aitor Navarro |

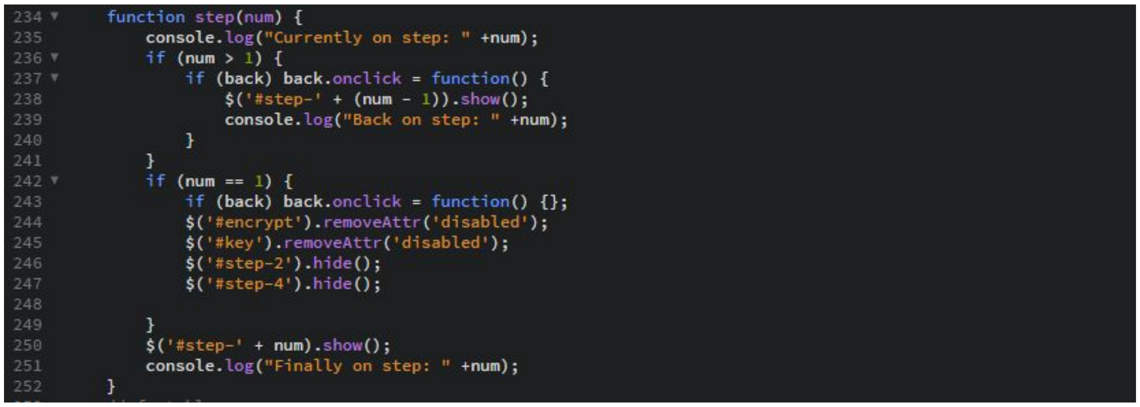


Chrome Extension

For the overall UI we use the Inspinia Admin theme.

Main script

In the main script, we toggle, hide and show the elements of out html files by calling our step function with different values.



```
function step(num) {
    console.log("Currently on step: " +num);
    if (num > 1) {
        if (back) back.onclick = function() {
            $('#step-' + (num - 1)).show();
            console.log("Back on step: " +num);
        }
    }
    if (num == 1) {
        if (back) back.onclick = function() {};
        $('#encrypt').removeAttr('disabled');
        $('#key').removeAttr('disabled');
        $('#step-2').hide();
        $('#step-4').hide();

    }
    $('#step-' + num).show();
    console.log("Finally on step: " +num);
}
```

Many basic functions like, URL regex checking, unit matching and event listening are necessary and therefore present. However, the most relevant functions for our work are the following :

- secureUpload:
    - We check we have a url pointing to our server and append the upload directory.
    - We generate an AJAX request with our encrypted file and its metadata.
- readFile:
    - We identify the scheme that has been selected.
    - We read from the file system in the format that best fits our scheme.
    - We call the specific library to perform the encryption of the file.
- prepAndSend (inside readFile):
    - We take the encrypted data in base64 format.
    - We generate an object with that data and the file's metadata and call secureUpload with that object.
- myFileList:
    - We generate an AJAX request for the file list.
    - We arrange the list obtained from the server within our UI, giving the user the options to decrypt and erase any of the files present.

WebCryptoAPI Wrapper

As we have explained, the introduction of the WebCrypto API is a highly significant advance for doing cryptography on the browser. However, due to the intense usage of complex Javascript resources lime promises among others, using it results in highly convoluted code.

For that reason we have developed a small wrapper that allows our main code to access the AES-related methods of the API with relatively easy and brief calls.

Below we are gonna include the code of our simplifying functions for key generation, encryption and decryption.

- Key Generation:

```
this.getKeyFromPassphrase = function(passphrase) {
    var iterations = 100000; //Could be more but that suffices and it's quick
    var salt = "Smthg as rndmd as pssbl,:*3u4813u58hadjifjiekf"
    var saltInBuffer = new TextEncoder().encode(salt);
    var passphraseInBuffer = new TextEncoder().encode(passphrase);
    //now we create the base key directly from the user's input and use that base key to derive a good one
    return window.crypto.subtle.importKey( "raw", passphraseInBuffer, {"name": "PBKDF2"}, false, ["deriveKey"]
    ).then(function (baseKey){
        return window.crypto.subtle.deriveKey( {"name" : "PBKDF2", "salt" : saltInBuffer, iterations,
        "hash" : "SHA-1" }, baseKey, {"name": "AES-CBC", "length": 256}, false, ["encrypt", "decrypt"]
        );
        console.log(baseKey);
    }).catch(function (err) {
        console.log ("Could not generate a valid key from passphrase: " + passphrase + " because: " + err);
    });

}
```

- Encryption:

```
this.encrypt = function(data, key) {
    var initializationVector = new Uint8Array(ivLen);
    crypto.getRandomValues(initializationVector);
    return crypto.subtle.encrypt( {"name": 'AES-CBC', iv: initializationVector}, key,data
    ).then(function(encrypted) {
        var ciphered = _me.joinIvAndData(initializationVector, new Uint8Array(encrypted))
        var base64 = StrFunc.bufferToBase64(ciphered).replace(/\-/g, '+').replace(/_/g, '\/');
        while (base64.length % 4) {
            base64 += '=';
        }
        return base64;
    });
};
```

- Decryption:

```
this.decrypt = function(buf, key) {
    var parts = this.separateIvFromData(buf);
    return crypto.subtle.decrypt({
        name: 'AES-CBC',
        iv: parts.iv
    }, key, parts.data).then(function(decrypted) {
        var base64 = StrFunc.bufferToBase64(new Uint8Array(decrypted)).replace(/\-/g, '+').replace(/_/g, '\/');
        while (base64.length % 4) {
            base64 += '=';
        }
        return base64;
    });
};
```

To show how these functions actually simplify the code of our main script we show now how the functions are actually used in our extension:

```
        if(window.scheme ===  "AESWC"){

            reader.onload = function(e){
                var ArrBufferView = new Uint8Array(e.target.result);
            console.log(ArrBufferView);
                webcrypto.getKeyFromPassphrase(pkey).then(function(key) {
                return webcrypto.encrypt(ArrBufferView, key)
            }).then(base64 => {prepAndSend(base64)}).catch( function(e) {
                    console.log("ERROR: " + e);
                    toastr.error('Failed to encrypt ' + file.name + '! Please try again.');
                    });
            }
```

As we can see, the asynchronous character of WebCryptoAPI implies that, even with the wrapper, operating with the API results in slightly convoluted code (lines highlighted in green).

Bibliography

[1] The biggest data breaches ever. http://money.cnn.com/2017/09/07/technology/business/biggest-breaches-ever/index.html.

[2] The 16 biggest data breaches of the 21st century. https://www.csoonline.com/article/2130877/data-breach/the-16-biggest-data-breaches-of-the-21st-century.html.

[3] Rules for the protection of personal data inside and outside the eu. https://ec.europa.eu/info/law/law-topic/data-protection_en.

[4] Cloud encryption: Bring your own key is no longer enough. https://www.globalbankingandfinance.com/cloud-encryption-bring-your-own-key-is-no-longer-enough. Accessed: 2017-06-30.

[5] Does byok mean 'barely your own keys'? https://ciphercloud.com/does-byok-mean-bring-your-own-keys.

[6] Usage in time of new privacy terminology. https://books.google.com/ngrams/graph?content=digital+privacy%2Conline+privacy%2Cinternet+privacy&case_insensitive=on&year_start=1957&year_end=2000&corpus=15&smoothing=0&share=&direct_url=t4%3B%2Cdigital%20privacy%3B%2Cc0%3B%2Cs0%3B%3BDigital%20Privacy%3B%2Cc0%3B%3Bdigital%20privacy%3B%2Cc0%3B.t4%3B%2Conline%20privacy%3B%2Cc0%3B%2Cs0%3B%3BOnline%20Privacy%3B%2Cc0%3B%3Bonline%20privacy%3B%2Cc0%3B%3BOnline%20privacy%3B%2Cc0%3B%3BONLINE%20PRIVACY%3B%2Cc0%3B.t4%3B%2Cinternet%20privacy%3B%2Cc0%3B%2Cs0%3B%3BInternet%20Privacy%3B%2Cc0%3B%3BInternet%20privacy%3B%2Cc0%3B%3BINTERNET%20PRIVACY%3B%2Cc0%3B%3Binternet%20privacy%3B%2Cc0.

[7] Daniel Osterwalder. Trust through evaluation and certification? *Social Science Computer Review*, 19(1):32–46, 2001.

[8] Robert Gellman. Privacy in the clouds: risks to privacy and confidentiality from cloud computing. In *Proceedings of the World privacy forum,*, 2012.

[9] Dawei Sun, Guiran Chang, Lina Sun, and Xingwei Wang. Surveying and analyzing security, privacy and trust issues in cloud computing environments. *Procedia Engineering*, 15:2852–2856, 2011.

[10] Siani Pearson and Azzedine Benameur. Privacy, security and trust issues arising from cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 693–702. IEEE, 2010.

[11] Siani Pearson. Taking account of privacy when designing cloud computing services. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 44–52. IEEE Computer Society, 2009.

[12] Karyn Benson, Rafael Dowsley, and Hovav Shacham. Do you know where your cloud files are? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 73–82. ACM, 2011.

[13] Siani Pearson, Yun Shen, and Miranda Mowbray. A privacy manager for cloud computing. In *IEEE International Conference on Cloud Computing*, pages 90–106. Springer, 2009.

[14] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *Infocom, 2010 proceedings ieee*, pages 1–9. Ieee, 2010.

[15] Ke Zeng and Ann Cavoukian. *Modelling cloud computing architecture without compromising privacy: A privacy by design approach.* Information and Privacy Commissioner of Ontario, 2010.

[16] Peter Schaar. Privacy by design. *Identity in the Information Society*, 3(2):267–274, 2010.

[17] Cis pet wiki. `https://cyberlaw.stanford.edu/wiki/index.php/PET`.

[18] Daniel J. Solove. A taxonomy of privacy. *University of Pennsylvania Law Review*, 2006.

[19] *Intelligence and Information Policy for National Security: Key Terms and Concepts.* Rowman and Littlefield, 2016.

[20] AJ Paverd, Andrew Martin, and Ian Brown. Modelling and automatically analysing privacy properties for honest-but-curious adversaries.

[21] ”Shafi Goldwasser and Silvio Micali”. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[22] Claude E Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949.

[23] Fauzan Mirza. Block ciphers and cryptanalysis. *Royal Holloway University of London, Department of Mathematics, England*, 1998.

[24] James H Ellis. The history of non-secret encryption. *Cryptologia*, 23(3):267–273, 1999.

[25] James H Ellis. The possibility of non-secret encryption. In *British Communications-Electronics Security Group (CESG*. Citeseer, 1970.

[26] Clifford C Cocks. A note on non-secret encryption. *CESG Memo*, 1973.

[27] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[28] M.E. Hellman, B.W. Diffie, and R.C. Merkle. Cryptographic apparatus and method, April 29 1977. US Patent 4,200,770.

[29] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[30] ECRYPT II. Final report on main computational assumptions in cryptography. *ECRYPT 2013*.

[31] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[32] Pascal Paillier et al. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, volume 99, pages 223–238. Springer, 1999.

[33] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[34] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, volume 3378, pages 325–341. Springer, 2005.

[35] David Mandell Freeman. Homomorphic encryption and the bgn cryptosystem. 2011.

[36] Jake Loftus, Alexander May, Nigel P Smart, and Frederik Vercauteren. On cca-secure somewhat homomorphic encryption. In *Selected Areas in Cryptography*, volume 7118, pages 55–72. Springer.

[37] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. A guide to fully homomorphic encryption.

[38] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford University, 2009.

[39] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[40] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[41] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multi-party computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[42] Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1518–1529. ACM, 2015.

[43] Protect Private Data. Functional encryption: beyond public key cryptography. 2008.

[44] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. *Theory of Cryptography*, pages 253–273, 2011.

[45] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001*, pages 213–229. Springer, 2001.

[46] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[47] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Advances in Cryptology–EUROCRYPT 2008*, pages 146–162, 2008.

[48] Michael Brightwell and H Smith. Using datatype-preserving encryption to enhance data warehouse security. In *20th National Information Systems Security Conference Proceedings (NISSC)*, pages 141–149, 1997.

[49] John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In *Cryptographers' Track at the RSA Conference*, pages 114–130. Springer, 2002.

[50] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In *Selected Areas in Cryptography*, volume 5867, pages 295–312. Springer, 2009.

[51] Mor Weiss, Boris Rozenberg, and Muhammad Barham. Practical solutions for format-preserving encryption. *arXiv preprint arXiv:1506.04113*, 2015.

[52] Daniel Luchaup, Kevin P Dyer, Somesh Jha, Thomas Ristenpart, and Thomas Shrimpton. Libfte: A toolkit for constructing practical, format-abiding encryption schemes. In *USENIX Security Symposium*, pages 877–891, 2014.

[53] Morris Dworkin. Recommendation for block cipher modes of operation: methods for formatpreserving encryption. *NIST Special Publication*, 800:38G.

[54] Gürkan Bebek. Anti-tamper database research: Inference control techniques. *Technical Report EECS 433 Final Report*, 2002.

[55] Gultekin Özsoyoglu, David A Singer, and Sun S Chung. Anti-tamper databases: Querying encrypted databases. In *DBSec*, pages 133–146, 2003.

[56] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM, 2004.

[57] Alexandra Boldyreva, Nathan Chenette, Younho Lee, Adam O'neill, et al. Order-preserving symmetric encryption. In *Eurocrypt*, volume 5479, pages 224–241. Springer, 2009.

[58] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[59] Raluca Ada Popa, Frank H Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 463–477. IEEE, 2013.

[60] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[61] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(1):5–20, 2018.

[62] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K Cunningham. Sok: Cryptographically protected database search. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 172–191. IEEE, 2017.

[63] Mark Houpt. Is there a difference between data in transit and data in motion? Web, September 2016.

[64] Thomas Ptacek. "javascript cryptography considered harmful, 2011.

[65] T Arcieri. Whats wrong with in-browser cryptography, 2013.

[66] Brendan McMillion. A criticism of javascript cryptography, 2017.

[67] Comparing performance of javascript cryptography libraries. `https://medium.com/@encryb/comparing-performance-of-javascript-cryptography-libraries-42fb138116f3`, June 2015.

[68] Language binding. `https://en.wikipedia.org/wiki/Language_binding`, October 2005.

[69] Language binding. `https://en.wikipedia.org/wiki/Use_case_diagram`, February 2012.

[70] Use case diagram. `https://msdn.microsoft.com/en-us/library/dd409432.aspx`, 2010.

[71] Spar(security and privacy assurance research) testing and assessment framework. `https://github.com/mit-ll/SPARTA`, 2015.

[72] Daniel S Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. Pope: Partial order preserving encoding. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1142. ACM, 2016.

List of Figures