PREaaS approach for secure file sharing in the cloud

Anass Sbai, Cyril Drocourt, Gilles Dequen

Université de Picardie Jules Vernes,
Laboratoire MIS - UniLaSalle / ESIEE 14 Quai de la Somme, 80080 Amiens, France
{anass.sbai,cyril.drocourt,gilles.dequen}@u-picardie.fr

Résumé
It is essential today for a company to store its data in an encrypted way when it uses Cloud Computing. However, the manipulation of this encrypted data remains complex, and it is very difficult in this case to be able to share the encrypted data between different users. One of the solutions for sharing encrypted data is to use PRE (Proxy Re-Encryption) which allows both the re-encryption of the data, but also the delegation of this operation by a third party via the use of a specific key. In this article, we propose a solution for sharing encrypted files between users that uses a classic storage system in the Cloud and PRE (Proxy Re-Encryption). We present an improvement of an existing PRE algorithm by applying it to elliptical curves in order to improve its performance. Finally, we implement this architecture in the form of a cloud service called PREaaS (PRE as a Service) which allows this mechanism to be used on demand with an API.

Mots-clés : Cloud data privacy, Proxy re-encryption, Encrypted file sharing, SECaaS.

---

1. Introduction

The use of encryption in Cloud Computing is now a must and a reality. The different actors of the field as well as the companies have understood the important stakes of data security. However, the systems used are often limited to general technologies based on TLS flow encryption and authentication system. Unfortunately, there are only a few solutions for storing encrypted data, and they are often tied to a user license or a company. Moreover, these solutions do not offer simple mechanisms for sharing encrypted data.
To this day, cloud storage remains restrictive in terms of confidentiality. In fact, the best-known providers (e.g. GoogleDrive, Dropbox, OVH ...) do not ensure the total confidentiality of their customers' data : either the data is encrypted by a key known by the Cloud Service Provider (CSP), or it is stored in clear. These CSPs are supposed to be considered honest but curious entities. However, they are considered entirely trustworthy entities. This is a real security issue, since there is a risk that the data will be given or even sold to third parties. In order to ensure confidentiality with respect to the CSP, the data must be stored encrypted and accessible only by authorized persons.
A naive solution to the problem of sharing encrypted data would be to use classical encryption techniques (symmetric or asymmetric) and to share the decryption key with the entities designated by the data owner.

Concerning symmetric encryption, it cannot be used alone since the same key is used for encryption and decryption. This implies that this same key is pre-shared between the data owner and the different entities to which we want to give access rights.
For public key cryptography, there are several solutions :

— All the recipients know the private key related to the public key of encryption, the problem is then identical to symmetrical encryption,
— Each data is encrypted with the public key of the recipient. The problem is that the users who should have the right to access the encrypted information are not necessarily known in advance.

Therefore, the only possibility is that this data is encrypted under a public key controlled by the data owner, which implies that the data owner must decrypt the data and then encrypt it with the user's own key. However, this decryption and encryption solution requires the data owner to be available and online to re-encrypt the data if necessary, which prohibits asynchronous management of the process. It is therefore extremely inefficient. The problem becomes increasingly complex when we consider multiple data and various entities.
The best solution would be to allow multiple people to decrypt the data without having to go through decryption/encryption steps or without sharing one's private or secret key. We consider it important to give the definition and the state of the art of the PRE so that the reader can better understand our choice. We also present in this paper, the architecture chosen for our solution as well as the implementation of the PRE.

2. Related Works

Several works aiming to protect privacy have emerged. We find for example CryptDB [1], ESPRESSO [2]. These solutions provide encryption services to maintain confidentiality. Nevertheless, each solution has its limitations. For example, CryptDB cannot be used for no SQL databases or file systems and the complexity of key management increases with the number of users. ESPRESSO uses only symmetric encryption and requires trusting a third party or CSPs to provide encryption and key management. Regarding data sharing, both solutions must go through decryption/encryption process.
This data sharing problem has been solved by various solutions in the literature, starting with broadcast encryption designed by [3]. In this case, each user can access the data independently of the others. This requires knowledge at the time of encryption of who will have the privilege to access the data. Another similar approach, introduced by [4], is attribute-based encryption (ABE). Inspired by D Boneh's work on identity-based encryption (IBE) schemes, their idea was to create a new type of IBE system [5] to combine encryption and access control. In this case, access privileges are not addressed to a set of users but only to users with a specific number of attributes. Unfortunately, this does not allow selective sharing.
As an alternative to these solutions, we choose to use the proxy re-encryption proxy (PRE). Thus, we will be able to transfer decryption rights to specific entities. The proxy re-encryption (PRE) is a crypto-system that allows to transform encrypted messages from an entity into messages that can be decrypted by an entity B . We then find the three main actors :

- The delegator (Alice) : noted "a" is the entity owner of the data which delegates the rights of decryption to another entity, namely the delegate, by creating a re-encryption key sent to the re-encryption proxy.
- The delegate (Bob) : noted "b" is the entity to which we granted the decryption rights of the encrypted messages which are not intended for him at the origin. These encrypted messages must be re-encrypted by the re-encryption key which is the object of an access

authorization for the delegate.
- The proxy : this entity manages the process of re-encrypting encrypted messages, originally intended for the delegate, into messages that can be decrypted by the delegate. The re-encryption key used by the delegate during the transformation process must not allow any disclosure of information to the proxy.

The first PRE was designed by BBS (Blaze, Bleum and Strauss) [6] based on the ElGamal asymmetric encryption system. The authors show that it is possible to incorporate a substitution key to re-encrypt an already encrypted message without compromising it. The BBS proposal is a very elegant solution and it allows to keep the same encryption and decryption algorithms as ElGamal. It also allows decrypting encrypted or re-encrypted messages with the same decryption function. This is due to the re-encryption process. Indeed, the latter transforms Alice's encrypted messages with ElGamal while keeping the same distribution for the re-encrypted messages intended for Bob. Thus, we cannot distinguish between an encrypted and a re-encrypted message. [7] formalize the design of re-encryption proxies by classifying these systems into two types : unidirectional and bidirectional.

[8] gives a more formal definition of PRE and concretely defines these properties such as :

— *Unidirectional* : Delegation of decryption rights from Alice to Bob does not allow Alice to decrypt Bob's cipher.
— *Non-interactive* : The re-encryption key can be generated by Alice without interacting with Bob, and thus using only Bob's public key.
— *Transparent* : Or invisible, meaning that the delegate can not distinguish between an encrypted message and a re-encrypted message.
— *Key-optimal* : The size of Bob's secret storage must remain unchanged, no matter how many delegations he accepts.
— *Original access* : The sender can decrypt any re-encrypted message which he was originally the owner.
— *Collusion-safe* : If the proxy and Bob collude, they shouldn't get Alice's secret key.
— *Non-transitive* : The proxy can not re-delegate re-encryption rights. (e.g from $\mathsf{Rk}_{a\to b}$ and $\mathsf{Rk}_{b\to c}$ the proxy can not calculate $\mathsf{Rk}_{a\to c}$)
— *Non-transferable* : The proxy and delegates can not redefine decryption rights. (e.g from $\mathsf{Rk}_{a\to b}$ and Pk_c and Sk_b we can not calculate $\mathsf{Rk}_{a\to c}$)
— *Temporary* : Bob can decipher the messages received from Alice only at a certain point in time.

The earliest studies [7] [8] focus more on creating an PRE that is unidirectional. These proposals provide security against IND-CPA attacks and are not resistant to collusion. [9] then propose the first PRE scheme secure against IND-CCA attacks where they prove the security of their scheme based on universal composability [10]. Their construction is bidirectional, uses pairing, and is proven secure in the random oracle model. The authors state that it is recommended to have a system which is proven secure in the standard model while avoiding pairing for efficiency reasons, whether for unidirectional or bidirectional systems.

The open problem presented by Canetti et al. has been addressed in [11]. The authors manage to solve a part of this open problem which concerns the construction of a secure PRE against CCA attacks without pairing. Their PRE is bidirectional and its is mainly based on ElGamal and Schnorr signature, which implies the use of random oracles. Based on [11], [12] proposed the first unidirectional PRE IND-CCA secure without pairings and thus relying only on ElGamal and Schnorr signature. [13] find a flaw in the security proof of Chow's construction and propose to fix it. Their construction is also unidirectional IND-CCA secure in the random oracle model and does not rely on pairings.

Since the appearance of proxy re-encryption, few applications make use of them. We can nevertheless mention Skycryptor[14] and Nucypher[15]. Skycryptor is a security solution as a service for file sharing. Basically, the proposed solution uses a unique symmetric key for each file to be encrypted with AES and then encrypts the secret key with the user's asymmetric public key generated through the PRE algorithm. The solution is a dedicated machine software and is now marketed under the name BeSafe. Each user's machine has its own key pair and re-encryption is used to share files between different devices or users. Most importantly, users must install the BeSafe software and use it to encrypt data.

3. Our Contribution

3.1. File Sharing

The goal of our system is to develop a CSP (Content Service Provider) application, in SaaS (Software As A Service) dedicated to storage. It should make it possible to replace proprietary solutions such as GDrive, DropBox, ICloud, etc. However, the goal is not to implement a new file system but to use existing standards that allow remote access.

Our application will have to use the classic organization of a file system, remain compatible with this model, and add additional information to solve our goal, without disturbing normal operation.

The objectives of the system are :

- to encrypt all of the stored information,
- to not expose the data by default to a person other than the owner of the data,
- to allow the sharing of encrypted data,

The additional constraints that we have defined :

- to have a quick solution in terms of use,
- to allow the sharing of a directory with another user, without copying it,

To solve the first constraint and the first objective, it is necessary to use a secret key algorithm. In addition, it is necessary to :

- encrypt each file independently, and therefore to generate a unique secret key to encrypt each file,
- not store the encryption key in the file itself, in order to simplify the organization of the files and access to them as well as to the encryption keys,

Sharing a directory requires access to all of the secret keys associated with all the encrypted files in the directory when the recipient user requests access. It is at this stage that asymmetric encryption is needed to encrypt the secret keys.

We have three solutions for storing the secret encryption keys associated with each file :

- Build an index file that contains all the secret keys and encrypt it with the owner's public key : this solution requires decrypting the entire index file to access each secret key, which is not practical ,
- Build an index file that contains all the secret keys, which will have been previously encrypted with the owner's public key : this always requires retrieving the entire file for each transaction,
- To create a specific file for each secret file containing the associated secret key, and encrypt this file with the owner's public key : It is this last solution that was chosen which allows finer granularity, because it also allows to process only the individually requested encrypted files.

In order to allow the sharing of directories, it is necessary that :

- our CSP has a method to transform encrypted data on demand,

— sharing a directory does not allow the recipient to access other data,
— sharing a directory does not allow the CSP to access other data,

The last point is essential in terms of security. Indeed, even in the event of a successful attack on the CSP, there should be no possibility of accessing data that the owner has not shared. Therefore, the only possible solution is not to use the owner's key pair directly, but for each directory to have a dedicated public / private key pair. Thus, as shown in "Fig. 1", for a file F1 located in a directory D1, a unique secret key K_{F1} will be generated to encrypt the file. This key will itself be encrypted with the public key of the Pk_{D1} directory.

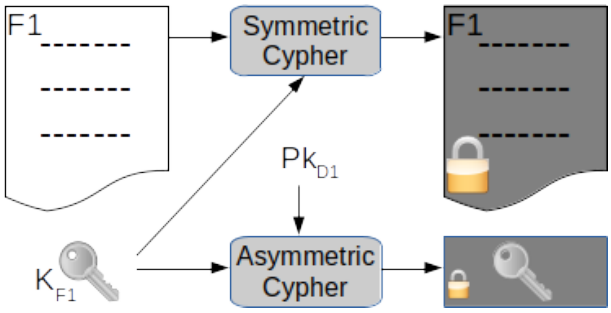

FIGURE 1 – Encryped file and secret key.

Now we have to solve another problem, which is to allow sharing of a directory to make the files in it accessible to the recipient. We must therefore make the secret keys available to the recipient, without providing the private key of the directory. It is therefore necessary to transform information encrypted with the public key of the directory, so that it can be decrypted with the private key of the recipient. This transformation is possible using a Proxy Re-Encryption (PRE). In this way, when the recipient requests access to a file that is shared, the CSP transforms the secret encryption key associated with the file using a re-encryption key to make it accessible to the recipient using the PRE. We propose to use PRE as a data sharing service in multi-cloud systems called PREaaS [16]. The PREaaS, has the advantage to only handles encrypted data and, public keys and re-encryption keys. It does not affect confidentiality in any way, even if the CSPs or users are corrupted and collude with the PREaaS. This is guaranteed by careful selection of the algorithms used by the service. The potential of cloud computing would take care of the consumption of re-encryption time.

As we can see in "Fig. 2", the application usage is as follows :

— Each user has his own key pair generated when creating the account on the CSP, for user A : Pk_a/Sk_a,
— (1) User A creates a directory D1, a corresponding asymmetric key pair is generated on the client side (Pk_{D1}/Sk_{D1}),
— (2) User A creates a file F1, a secret encryption key K_{F1} is generated, the file is encrypted with this key,
— (3) The secret key is itself encrypted with the public key Pk_{D1} of the directory and sent to the server to be placed in an index file I_{F1},
— (4) The owner wants to share a directory with user B, he generates on the client side a re-encryption key $Rk_{D1\to B}$ allowing to transform the data encrypted for A into data encrypted for B, this key is sent to the CSP,
— (5) User B requests access to the file F1 shared by A,

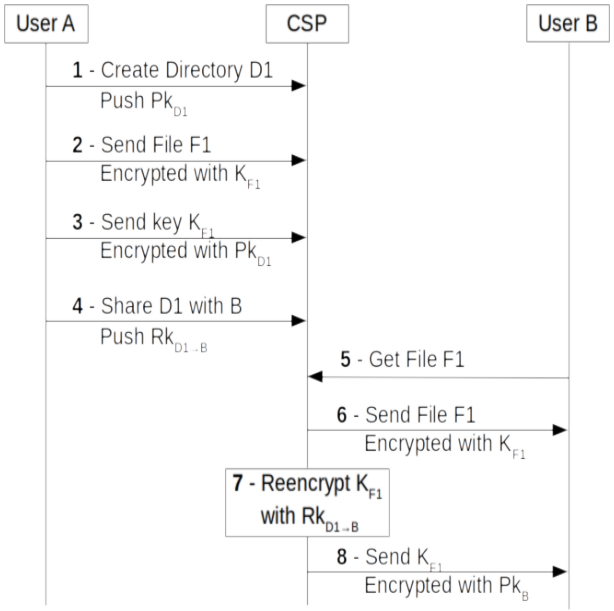

FIGURE 2 – Interactions between Users and CSP.

— (6) The CSP sends him the requested file encrypted with the key K_{F1},
— (7) The CSP re-encrypts the index file I_{F1} containing the secret key K_F1 using the re-encryption key $Rk_{D1 \to B}$,
— (8) The CSP sends the index file I_{F1} containing the secret key K_{F1} encrypted with the key Pk_b to user B,
— User B who has the private key Sk_b can therefore decrypt the file I_{F1} containing the secret key, then decrypt the file F1,

In order to securely store the information specific to each user (personal key pair and key pairs associated with each directory), a secret key derived from the user's password is generated on the client side which allows this information to be encrypted before sending it to the CSP. In this way, the CSP does not manipulate any personal user data, only encrypted data.

As we will see in the part concerning the implementation, our CSP is therefore composed of two remote services, the PREaaS and the file server. In addition, the main component of the application is located on the client side, and it is executed directly in the internet browser.

3.2. Proxy Re-Encryption

Usually, a PRE scheme can be defined as a tuple ζ : {Setup, KeyGen, ReKeyGen, ξ^{asym}, ReEnc, Dec}. Where :

— $Setup(1^k) = params$: takes as input a security parameter k and generates the system parameters which generally define the recommended length of messages and keys.
— $KeyGen(params) = (Pk, Sk)$: is the function that generates the public/private key pair Pk and Sk respectively.
— $RekeyGen(Sk_a, Pk_b) = Rk_{a \to b}$: in the case of the one-way ERP (defined in 2.2.3), it is necessary to enter the private key of a and the public key of b to generate the re-encryption key.
— $\xi^{asym}_{Pk_a}(M) = C_a$: is the encryption function. It takes as input a plaintext message M and a public key Pk .

— $ReEnc(C_a, Rk_{a\to b}) = C_b$: is the re-encryption function. It takes as input an encrypted message and a re-encryption key.
— $Dec(C, Sk) = M$: is the decryption function. It takes as input an encrypted or re-encrypted message and returns the corresponding plaintext message.

In some cases, we can find two other functions used for encryption and decryption, so that the encrypted message created by this encryption function (also called non-transformable encryption) cannot be re-encrypted and only the owner of the private key can decrypt it.

This part presents the PRE algorithm that we use for file sharing. Thanks to this survey [17], we can see the computational comparison of PRE schemes. By keeping only the schemes that are CCA secure we can conclude that Chow's algorithm is the more efficient among them. In [13] the authors find a flaw in the security proof of Chow's construction and propose to fix it.

Thus, we chose [13] because it reaches IND-CCA security and also because of its efficiency which is due to the fact that it does not require pairing. Most importantly, the scheme is unidirectional and collusion resistant.

Two instances have been implemented, the first one uses a generic group of prime order, and the second one uses a group of point on elliptic curve. We first implemented the algorithm by simply using a generic group of prime order. We quickly realized that this solution was unthinkable. Indeed, the time necessary for the use of the cryptographic primitives make the system unusable in real time.

We then studied the other possibilities as well as the other existing PRE algorithms but without success. We then had the idea to change the method, and adapt the algorithm of [13] to use it with elliptic curves.

The main difference between the two instantiations is that the elliptic curve point group is an additive group. Thus, all exponentiations are transformed into scalar multiplications. Also, the hash functions had to be adapted in order to generate elliptic curve points. This algorithm could be instanciated using any group where the computational Diffie-Hellman problem is known to be hard. For efficiency reason we use elliptic curves. We will see later in our implementation that the lenght of keys and ciphers will be much reduced as well as the time consumption. We give a formal version of the Selvi's algorithm with ECC below :

— $Setup(1^k)$: for 128-bit, choose a prime p of 256-bit length and an elliptic curve $\mathbb{E}_{\mathbb{F}_p}$ such that $a = p - 3$. $G = (x_G, y_G)$, a point on the curve, known as the base point with order n. The elliptic curve equation is then : $y^2 = x^3 - 3x + b \bmod (p)$. (You can find the exact values of NIST p-256 curve parameters in [18]). Choose five hash functions : $H_1 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \to \mathbb{Z}^*_p, H_2 : \mathbb{E}_{\mathbb{F}_p} \to \{0,1\}^{l_0+l_1}, H_3 : \{0,1\}* \to \mathbb{Z}^*_p, H_4 : \mathbb{E}_{\mathbb{F}_p} \to \mathbb{Z}^*_p, H_5 : \mathbb{E}^4_{\mathbb{F}_p} \times \{0,1\}^{l_0+l_1} \to E_{Fp}$. The message space M is $\{0,1\}^{l_0}$, where $l_0 = l_1 = k$. $params = (p, \mathbb{E}_{\mathbb{F}_p}, G, H_1, H_2, H_3, H_4, l_0, l_1)$.
— $KeyGen(params)$:
  — Pick $x_{a,1}, x_{a,2} \xleftarrow{\$} \mathbb{Z}^*_p$.
  — Compute $Pk_{a,1} = x_{a,1}.G$, $Pk_{a,2} = x_{a,2}.G$
  — Return $Sk_a = (x_{a,1}, x_{a,2})$ & $Pk_a = (Pk_{a,1}, Pk_{a,2})$.
— $RekeyGen(Sk_a, Pk_a, Pk_b)$:
  — Pick $h \xleftarrow{\$} \{0,1\}^{l_0}, \pi \xleftarrow{\$} \{0,1\}^{l_1}$.
  — Compute $r = H_1(h, \pi), V = r.Pk_{b,2}$ and $W = H_2(r.G) \oplus (h||\pi)$.
  — Define $rk = \dfrac{h}{x_{a,1} H_4(Pk_{a,2}) + x_{a,2}}$
  — Return $Rk_{a\to b} = (rk, V, W)$.
— $\xi^{asym}_{Pk_a}(m)$:

- Pick $u \xleftarrow{\$} \mathbb{Z}^*_p, w \xleftarrow{\$} \{0,1\}^{l_1}$.
- Compute $D = u.[(H_4(Pk_{b,2}).Pk_{a,1}) * Pk_{a,2}]$, $r = H_1(m,w)$ & $E = r.[(H_4(Pk_{b,2}).Pk_{a,1}) * Pk_{a,2}]$.
- Compute $F = H_2(r.G) \oplus (m||w)$.
- Compute $\overline{D} = u.H_5(Pk_{a,1}, Pk_{a,2}, D, E, F)$ & $\overline{E} = r.H_5(Pk_{a,1}, Pk_{a,2}, D, E, F)$.
- Compute $s = u + r \times H_3(D, \overline{E}, F) \bmod(p)$.
- Return $C_a = (D, \overline{E}, F, s)$.

— $ReEnc(C_a, Pk_a, Pk_b, Rk_{a\to b})$:

- Compute D and $\overline{D}$:
  $D = s.[(H_4(Pk_{b,2}).Pk_{a,1})*Pk_{a,2}]*(H_3(E,\overline{E},F).E)^{-1}$
  $\overline{D} = (s.H_5(Pk_{a,1}, Pk_{a,2}, D, E, F))*(H_3(E,\overline{E},F).\overline{E})^{-1} = u.[H_5(Pk_{a,1}, Pk_{a,2}, D, E, F)]$
- Check if :
  $s.[(H_4(Pk_{b,2}).Pk_{a,1})*Pk_{a,2}]=D*(H_3(E,\overline{E},F).E)$ (1)
  $s.[H_5(Pk_{a,1}, Pk_{a,2}, D, E, F)]=\overline{D}*(H_3(E,\overline{E},F).E)$ (2)
- If the above check fail return +. Else, compute $E' = E^{rk} = (r \times h).G$
- Return $C'_b = (E', F, V, W)$.

— $Dec(C_a, Pk_a, Sk_a)$:

(Original CipherText)

- Compute D and $\overline{D}$. Then check if (1) & (2) holds.
- If the condition hold, compute $(m||w) = F \oplus H_2(\frac{1}{x_{a,1}H_4(Pk_{a,2} + x_{a,2})}.E)$
- If : $E = H_1(m,w).[(H_4(Pk_{b,2}).Pk_{a,1})*Pk_{a,2}]$ and $\overline{E} = H_1(m,w).[H_5(Pk_{a,1}, Pk_{a,2}, D, E, F)]$ return m , else return +.

(Transformed CipherText)

- Compute $(h||\pi) = W \oplus H_2(\frac{1}{Sk_{a,2}}.V)$.
- Extract $(m||w) = F \oplus H_2(\frac{1}{h}.E')$.
- If : $V=H_1(h,\pi).Pk_{a,2}$ and $E'=(h \times H_1(m,w)).G$ return m , else return +.

As far as hash functions are concerned, the only particularity is the mapping into a curve point for the H_5 function. A simple hash function may not be the best idea for this problem. Let's assume that we can use SHA-3 and that the fingerprint generated through this function will be the x coordinate value. Through this value, we can use the equation of the curve to calculate the value of the y coordinate which will be a square root. We will then have two possible values for y. We have to pick one of the two values at each new call to this function. Then, when it comes time to verify, we need to call the function twice and check which of the two results matches our curve point. A possible, but not very efficient, solution is to use SHA-3 to generate the y ordinate. We know that it corresponds to a single point of the curve and that a value is possible for the abscissa x. However, we will have to solve the equation of the curve and end up calculating the cube root, which is more expensive in terms of calculation. We used SHA-3 with a variant of the Koblitz method. However, the Koblitz method and its variant are not the best options in the literature for hashing into elliptic curve points.There are some more optimal and safer solutions allowing to do both the hash and the mapping to a curve point such as [19] and [20]. This problem has not been addressed in our work.

3.3. Implementation

We chose to use JavaScript as a core technology. And thus to be executed in the client side directly by navigator or even mobile devices without any software and also in the server side,

thanks to Nodejs. For the tests we used a 2,5 GHz intel core i7, with 16 GB RAM.

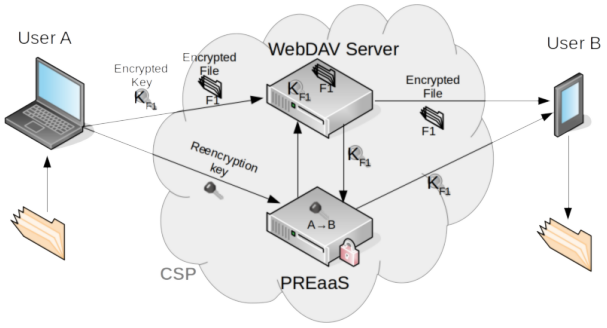

FIGURE 3 – Tasks distribution between different environments.

The purpose of the application is to use an existing network file system and it must be able to function as an additional layer of encryption. In this way, all storage systems could be used. As a first step, we decided to test our application using file sharing based on the WebDAV protocol. Indeed, this protocol is widely supported by a large number of client software, it is light, easy to use, and allows us to perform the operations we need :

- Create directories,
- Send files,
- Download files,
- Delete remote elements,

Since part of the application runs in JavaScript on the client, the majority of operations are performed by the client. It is also the client who downloads directly from the file server without going through the PREaaS. It just re-encrypts and transmits the URLs. We can see in "Fig. 3" the principle of interaction between the PREaaS and the WebDAV file server.

For the implementation of the Selvi's algorithm, two instance were tested. First we use a generic group with prime order length 3072-bit, and the second one using NIST Standard ECC p-256 [18] thanks to SJCL (Standford Javascript Crypto Library) [21]. Both correspond to the same security level that is 128-bit.

TABLE 1 – Computational efficiency of Selvi's algorithm in (ms)

Function	$\mathbb{F}_p$(ms)	$\mathbb{F}_p$(bits)	ECC(ms)	ECC(bits)
KeyGen	515	512 & 6144	68	512 & 512
ReKeyGen	502	3584	48	768
Encrypt	1000	6656	95	1024
ReEncrypt	967	6656	89	1024
Decrypt	979	–	78	–

Table 1. shows the time resources consumed by the different functions of Selvi's algorithm for both implementations. We measured the execution time of each of the PRE functions (Keygen,

ReKeyGen, ...) on a series of several executions for which we calculated an average.
Regarding the size corresponding to each function, it refers to the following elements :

— KeyGen : refers to the size of private keys $2|\mathbb{Z}_q|$ & public keys $2|\mathbb{G}|$
— ReKeyGen : refers to the size of re-encryption keys $|\mathbb{G}| + |\mathbb{Z}_q| + l_1 + l_2$
— Encrypt : refers to the size of original ciphertexts $2|\mathbb{G}| + |\mathbb{Z}_q + l_1 + l_2$
— ReEncrypt : refers to the size of re-encrypted ciphertexts $2|\mathbb{G}| + 2l_1 + 2l_2$

We can see that no operation is constraining in terms of time consumption if we use ECC. Indeed, we notice that the size has considerably decreased as well as the execution time have been reduced by a ratio of approximately 10 when using an elliptic curve. We can see also that encryption and re-encryption functions consume the most compared to key generation and decryption. In practice, the encryption and key generation functions are not often called. Re-encryption key generation depends on the number of delegations needed, but it is not yet time consuming. Instead, the re-encryption function is called for each new delegation and changed key. Having an independent service such as the PREaaS that does the re-encryption work is relieving.

4. Conclusion

In this article we present an original approach to solve the problem of sharing encrypted files. The foundation of this work remains the PRE, however, to date, no solution nor implementation for real-time use has been proposed in the literature. We address this essentially in three aspects :

— The modification of the SELVI algorithm in order to use elliptic curves and to obtain a 10-fold gain in speed,
— The use of Javascript as the implementation language, allowing a use in any mobile terminal with a simple Internet browser,
— The use of a PRE as a Service, allowing to deport the re-encryption procedures on a remote server, and to perform these operations in asynchronous mode.

Our CSP SaaS application uses standard protocols to store files (WebDAV) and to communicate (HTTP) . In order to not expose any data, they are encrypted by the browser before sending it to the server. Each file is encrypted with an independent secret key, which is itself encrypted with a public key specific to each directory. The encryption algorithm allows decryption rights to be delegated to another user, by generating a re-encryption key that is transmitted to the server. Our re-encryption application works as a service we call PREaaS.

Bibliographie

1. R. A. Popa, N. Zeldovich, and H. Balakrishnan, "Cryptdb : A practical encrypted relational dbms," 2011.
2. S. Kang, B. Veeravalli, and K. M. M. Aung, "Espresso : An encryption as a service for cloud storage systems," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, 2014, pp. 15–28.
3. A. Fiat and M. Naor, "Broadcast encryption," in *Annual International Cryptology Conference*. Springer, 1993, pp. 480–491.
4. A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 457–473.
5. D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Springer, 2001, pp. 213–229.

6. M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 127–144.
7. A.-A. Ivan and Y. Dodis, "Proxy cryptography revisited." in *NDSS*, 2003.
8. G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 1–30, 2006.
9. R. Canetti and S. Hohenberger, "Chosen-ciphertext secure proxy re-encryption," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 185–194.
10. R. Canetti, "Universally composable security : A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.
11. R. H. Deng, J. Weng, S. Liu, and K. Chen, "Chosen-ciphertext secure proxy re-encryption without pairings," in *International Conference on Cryptology and Network Security*. Springer, 2008, pp. 1–17.
12. S. S. Chow, J. Weng, Y. Yang, and R. H. Deng, "Efficient unidirectional proxy re-encryption," in *International Conference on Cryptology in Africa*. Springer, 2010, pp. 316–332.
13. S. S. D. Selvi, A. Paul, and C. Pandurangan, "A provably-secure unidirectional proxy re-encryption scheme without pairing in the random oracle model," in *International Conference on Cryptology and Network Security*. Springer, 2017, pp. 459–469.
14. A. Jivanyan, R. Yeghiazaryan, A. Darbinyan, and A. Manukyan, "Secure collaboration in public cloud storages," in *CYTED-RITOS International Workshop on Groupware*. Springer, 2015, pp. 190–197.
15. M. Egorov, D. Nuñez, and M. Wilkison, "Nucypher : A proxy re-encryption network to empower privacy in decentralized systems," 2018.
16. A. Sbai, C. Drocourt, and G. Dequen, "Pre as a service within smart grid cities," in *16th International Conference on Security and Cryptography*, 2019.
17. Z. Qin, H. Xiong, S. Wu, and J. Batamuliza, "A survey of proxy re-encryption for secure data sharing in cloud computing," *IEEE Transactions on Services Computing*, 2016.
18. S. Gueron and V. Krasnov, "Fast prime field elliptic-curve cryptography with 256-bit primes," *Journal of Cryptographic Engineering*, vol. 5, no. 2, pp. 141–151, 2015.
19. T. Icart, "How to hash into elliptic curves," in *Annual International Cryptology Conference*. Springer, 2009, pp. 303–316.
20. M. Tibouchi and T. Kim, "Improved elliptic curve hashing and point representation," *Designs, Codes and Cryptography*, vol. 82, no. 1-2, pp. 161–177, 2017.
21. E. Stark, M. Hamburg, and D. Boneh, "Stanford javascript crypto library," 2013.