Concordia University

INSE 6140

Middleware and Application Security

Puzzlr
Decentralized Secure Image Sharing App

Aniss Chohra (40001217)

Quentin Le Sceller (40002477)

Submitted to:
Professor Makan Pourzandi

April 14, 2016

Contents

List of Figures

List of Tables

1 Abstract

2 Introduction

As smartphones and tablets sales continue to rise enormously nowadays, the need for new mobile applications has obviously not ceased increasing. Developpers have achieved a huge progress in this fieled in the last decade, and still, there are many new and interesting ideas that need to be implemented in the near future.

Today, mobile applications are covering different palettes of services. Ranging from worldwide daily news, gaming and other entertainment services to social networking and educational services, and from sports, music, and financial services to lifestyle and many other categories.

Amongst these categories, photo and video applications are increasingly used these days. They allow their users to take pictures, record videos, and share them with their friends. SnapChat is a famous example of these, where the user takes an ephemeral or temporal pictures which will be sent to his/her friends. When the other user receives the pictures, he/she will be able to see it for only few seconds and then the image is completely removed.

However, if someone reads the SnapChat Terms of Service Licence, which unfortunately the majority of users do not, he/she will notice a major deceptive side of this kind of application:

"For all Services[...], you grant Snapchat a worldwide, royalty-free, sublicensable, and transferable license to host, store, use, display, reproduce, modify, adapt, edit, publish, and distribute the content."

As the above infers, when the user sends a picture to another one, this picture will be stored by the application's server on a specific database and will not be secure. This is a major problem, especially when people are sharing personal and intimate images.

In this project, we developped a mobile application for both mobile oprating systems: Android and IOS, in which we resolved the problem by using strong cryptographic primitives to provide data confidentiality and integrity during the whole exchange process. This solution provides secure communications between the clients and the server in order to securely exchange session keys which are used then to encrypt images shared between different clients of the application.

The paper will be organized as follows: in the third section, we will discuss some previous and related works on this subject. In the fourth section, we will present our threat model in which we will discuss some potential threats on the application. Section five will cover a brief overview on the cryptographic primitives and methods used during the developpement. Section six introduces our proposed architecture and how the application works. Finally, section seven will cover the three main implementations: Android, IOS, and the Server side.

3 Threat model

4 Cryptography

4.1 Client-Server Communications

For such mobile application, we need to be sure that the data exchanged between the server and the client cannnot be tampered with.

In order to secure the communications between the client and the server, we used TLS 1.2 with a strong preferred cipher suite. In our case, we chooses ECDHE for key exchange mechanism, RSA for the authentication, AES (128 bits) in GCM mode and SHA256 for the MAC. With OpenSSL enabled, we guarantee confidentiality, integrity and authentication between the client and the server.

Moreover, all the requests to the server are authenticated in order to prevent spam and denial-of-service (DOS) attacks.

4.2 Password Storage

All password are hashed on the server-side with bcrypt[1] using 10 rounds. It guarantees that all passwords on the database are protected.

4.3 Data Encryption

In this section, we are going to discuss some cryptographic primitives that we used in the developpement of our application in order to secure the data exchange between different participants and to provide confidentiality.

4.3.1 Image Encryption

For images encryption, we have chosen the famous and well-known block cipher cryptosystem: Advanced Encryption Standard (AES). As a reminder of how AES works, it takes as input an AES key (128, 192, 256 bits length) and an Initialization Vector(IV) of 16 bytes (128 bits) length and a plaintext of any size.

The algorithm starts first by dividing the plaintext into multiple blocks (each one has the same size as the key), this leads in general to the issue where the size of the plaintext is not a multiple of the size of the key. This is why we are obliged to pad the plaintext to make it multiple of the key size when the problem occurs.

[1] `http://bcrypt.sourceforge.net/`

But the question that someone might ask is: why did we choose to work with AES, why not with another algorithm like 3DES (Triple Data Encryption Standard) ? Well, the answer can be given as follows:

- AES is more secure and is less vulnerable to cryptanalysis unlike 3DES.
- AES supports larger key sizes than 3DES and thus larger block sizes. And this makes AES less vulnerable to attacks like *Birthday Paradox problem* than 3DES.
- AES is faster in both hardware and software.
- 3DES is breakable while AES is still unbreakable.
- AES uses substitution-permutation which are much more faster operations than *Feistel Networks* which are used by DES.

4.3.2 Key Encryption

Obviously, if Alice and Bob need to exchange an encrypted picture, they need first to securely exchange the session AES key. This is achieved by using the public key encryption (asymmetric encryption).

In this work, we used the well-known public cryptosystem: Rivest Shamir Adleman (RSA). When Alice and Bob register, they generate an RSA keypair, private key and public key. As their name infer, the public key is distributed to all the participants while the private key must be stored in a very secure manner and only its owner ought to know it.

In public key encryption schemes, the public key of your correspondant is used to encrypt the data. And when your correspondant receives this encrypted data, he will be the only one able to reverse the process with his/her private key.

The participant that wants to establish a session (let's say Alice) (send a picture) with the other will first generate the AES session key, then she will use Bob's public key to encrypt this AES key. In that way, only Bob can decrypt the message using his private key and thus the secure exchange of AES sessions keys is ensured. In this project, because RSA encryption is really consuming and takes much more time, we used the basic and minimal size for the size of the keypair which is 2048 bits (but it is still secure enough).

4.4 Data Integrity

As for data integrity, we used Message Authentication Codes (MAC). The process consists of generating a MAC key of 32 bytes long using the following algorithm: ***HmacSHA512***. Then a MAC tag is computed on the AES ciphertext and the IV using this key.

Table 1 briefly describes all the cryptographic primitives that we discussed above:

Symmetric Encryption	***Description***	***Size***
Key	Generated using a PBKDF2 (Password-Based Key Derivation Function 2) with 10000 iterations and a random salt.	32 bytes or 256 bits length.
IV	Generated using a PBKDF2 also with the same iterations and a random salt.	16 bytes or 128 bits length.
Mode	CBC (Cipher Block Chaining) which ensures the transmission of the randomness of the IV from the first block cipher to the rest of them.	each block has the same size of the key (256 bits).
Padding	PKCS5Padding which works as the following: • The number of bytes to be padded equals to: 8 - ((the number of bytes of the plaintext) mod 8). • All padded bytes have the same value as the number of bytes to be added.	From 1 to 8 bytes
Asymmetric Encryption	***Description***	***Size***
Private key	Stored securely on its owner device and knwon only by him/her and used for decryption.	2048 bits.
Public key	Distributed to all the other parties and used for encryption.	2048 bits.
Ciphertext	Computed only on the AES key, MAC key.	Depends on the *Modulus* size (always equal to its size).
MAC	***Description***	***Size***
Key	Generated using ***HmacSHA512*** algorithm specification.	256 bits.
tag	Computed only on the AES ciphertext and the IV.	256 bits.

Table 1: Cryptographic primitives.

5 General Architecture

As an architecture of our implementation, we used a semi-decentralized scheme where in each session, there are three major participating parties (figure 1).

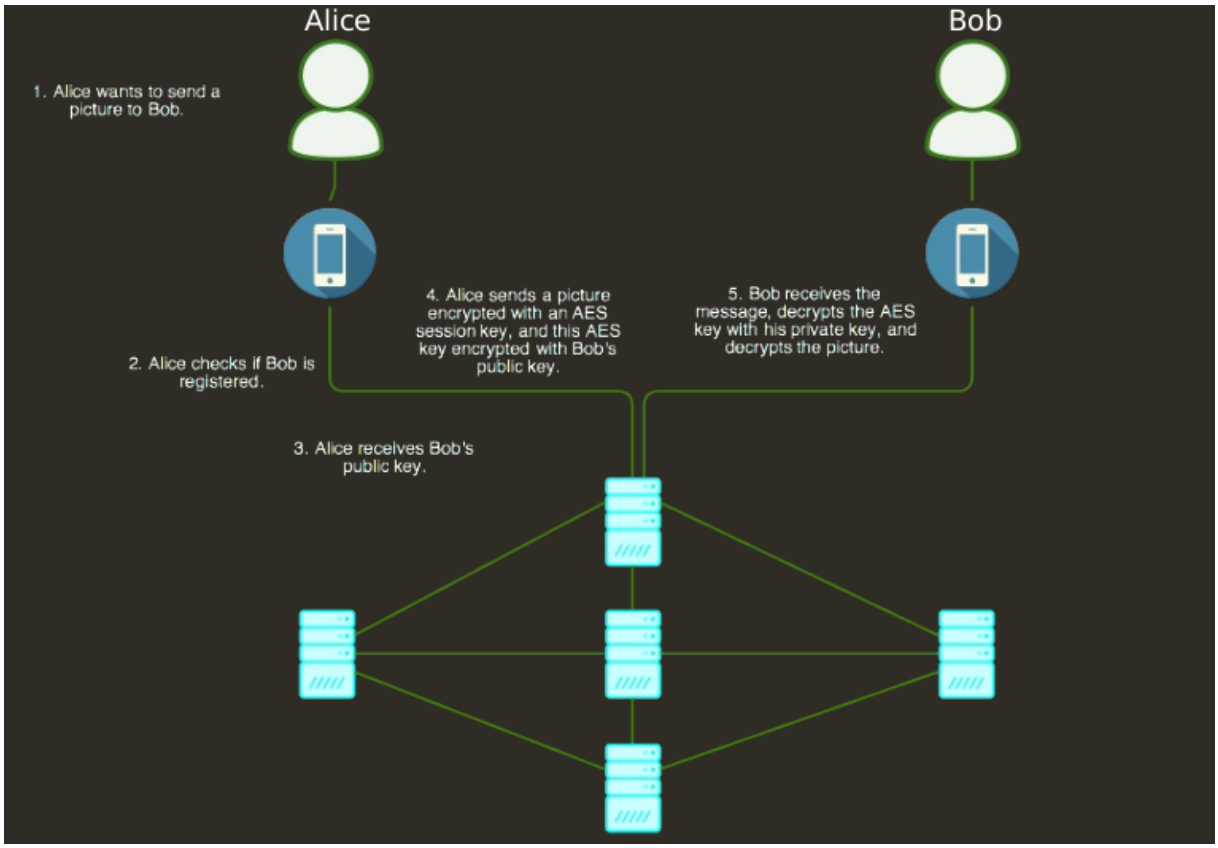

Figure 1: Puzzlr: General Architecture.

5.1 Registration Phase

When two clients want to register on the application, say Alice and Bob, they will both generate an RSA key pair. Their respective public keys are going then to be sent to the server to be stored on the database (figure 2).

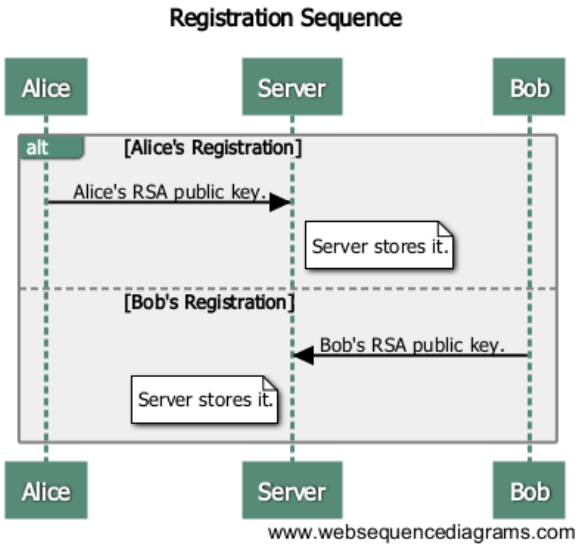

Figure 2: Registration phase Sequence Diagram.

5.2 Retrieval of Other Correspondant's Public key

If let's say Alice wants to send a message to Bob, she will first have to find his corresponding public key. To achieve that, she will send a request to the server which will query the database to check if Alice and Bob are both registered first, if that is the case, the server will then send Bob's public key to Alice (figure 3).

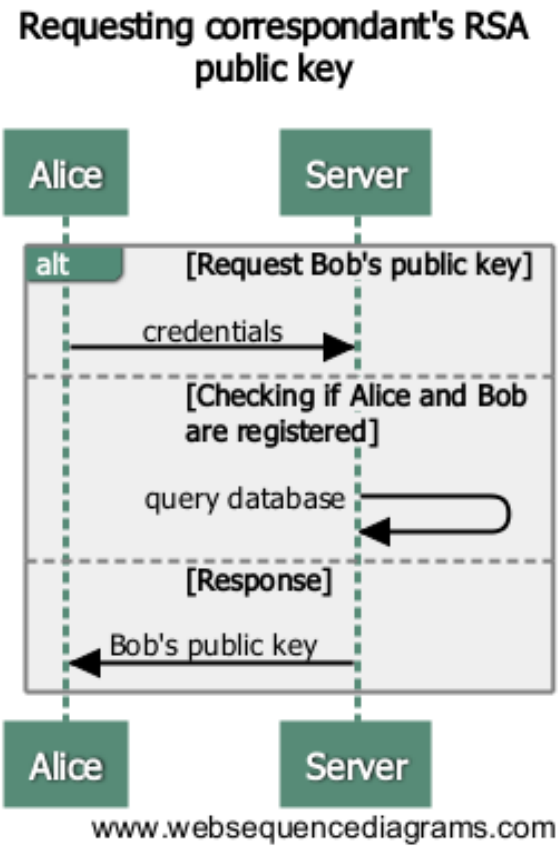

Figure 3: Requesting Correspondant's RSA public key Sequence Diagram.

5.3 Picture Sending and Receival

After retreiving Bob's public key from the server, Alice will now do the following steps (figure 4):

- Choose a picture to send.
- Generate a session key (AES) and a MAC key.
- Generate a random IV.
- Encrypt the AES key, the MAC key, and her username using Bob's public key (message 1).
- Encrypt the picture that she picked with the AES key and the IV (message 2).
- Generate a MAC tag using the MAC key on the second message (message 2) and the IV (message 3).
- Send the three messages to the server concatenated (message 1 + message 2 + message 3).

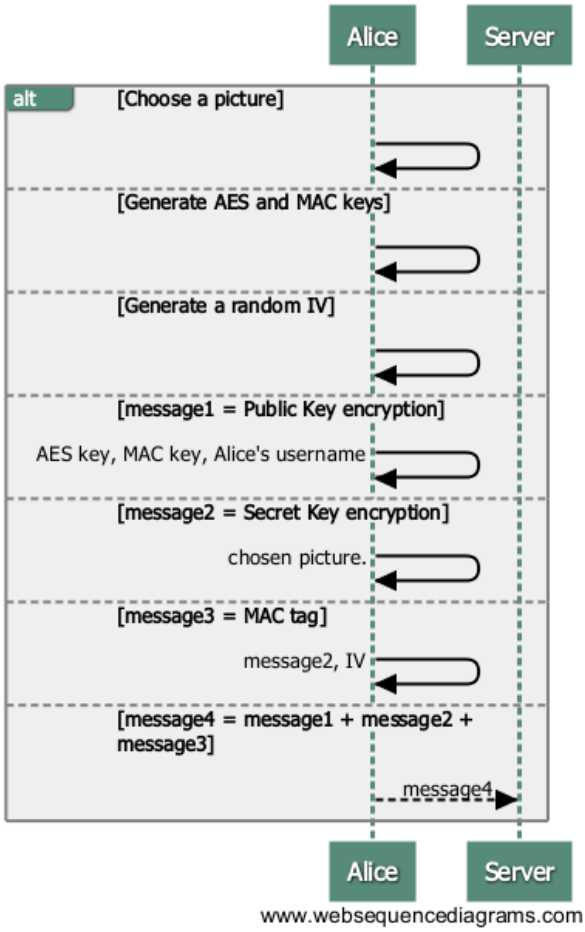

Figure 4: Picture Sending

On the other hand, when the server receives Alice's message, it will forward this message to Bob. Bob will receive the message and (figure 5):

- Recovers the AES key, MAC key, and Alice's username using his private key.
- Checks if the MAC tag received is valid by computing a new one on the received AES ciphertext and the IV, and then compares between them.
- Finally, recovers the picture using the recovered AES key and the IV.

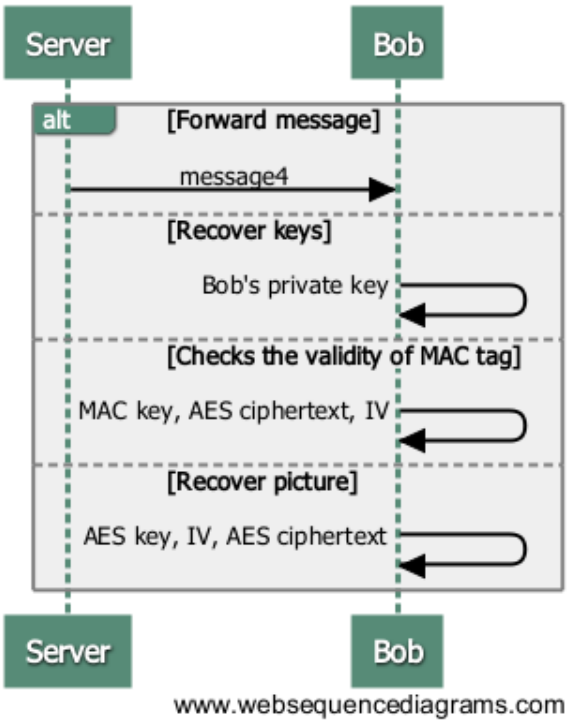

Figure 5: Picture Receival

6 Implementations

All of our implementations (server, Android client and iOS client) are available on GitHub (`https://github.com/quentinlesceller/Puzzlr`) under MIT License.

6.1 Server side

6.1.1 Requirements

6.1.2 Fabric

6.2 Client side

6.2.1 Android

The Android version is written in Java Programming Language alongside with some parts in XML (the Graphical User Interface). In this project we used Android Studio as an Integrated Developpement Environment (IDE). The application is compatible with almost all versions of Android.

The source code is open source and is available at this link along with the installation details:

https://github.com/aniss05/Puzzlr2

6.2.2 IOS