Puzzlr
Decentralized Secure Image Sharing App

Aniss Chohra

Quentin Le Sceller

May 5, 2016

Contents

List of Figures

List of Tables

1 Abstract

In a more and more connected world, privacy and control over our own data has become a right for every citizen. We believe that encryption need to be accessible for the real world and easy to use. In this pape,r we describe Puzzlr,a decentralized secure image sharing app for mobile devices. Puzzlr relies on strong cryptography and protect user anonymity.

2 Introduction

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 user 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 friends. When the other user receives the pictures, he will be able to see it for only few seconds and then the image is removed from his device.

However,most of the time, in the terms of services of these services we can find paragraph such as the following from Snapchat :

"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 maybe not securely. This is a major problem, especially when people are sharing personal and private images.

In this project, we developed a mobile application for Android and IOS, in which we tackle the problem of image sharing 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. Moreover, the backend rely on Fabric, a decentralized server in order to avoid government censorship and downtime.

The paper will be organized as follows: we first describe the general architecture of Puzzlr and how it works. Then we describe the cryptographic primitives and methods used. We also cover the three main implementations: Android, IOS, and the Server side. In the last part, we detail the future work.

3 General Architecture

In this part, we present a global overview over Puzzlr's architecture. We used a semi-decentralized scheme where in each session, there are three major participating parties (figure 1).

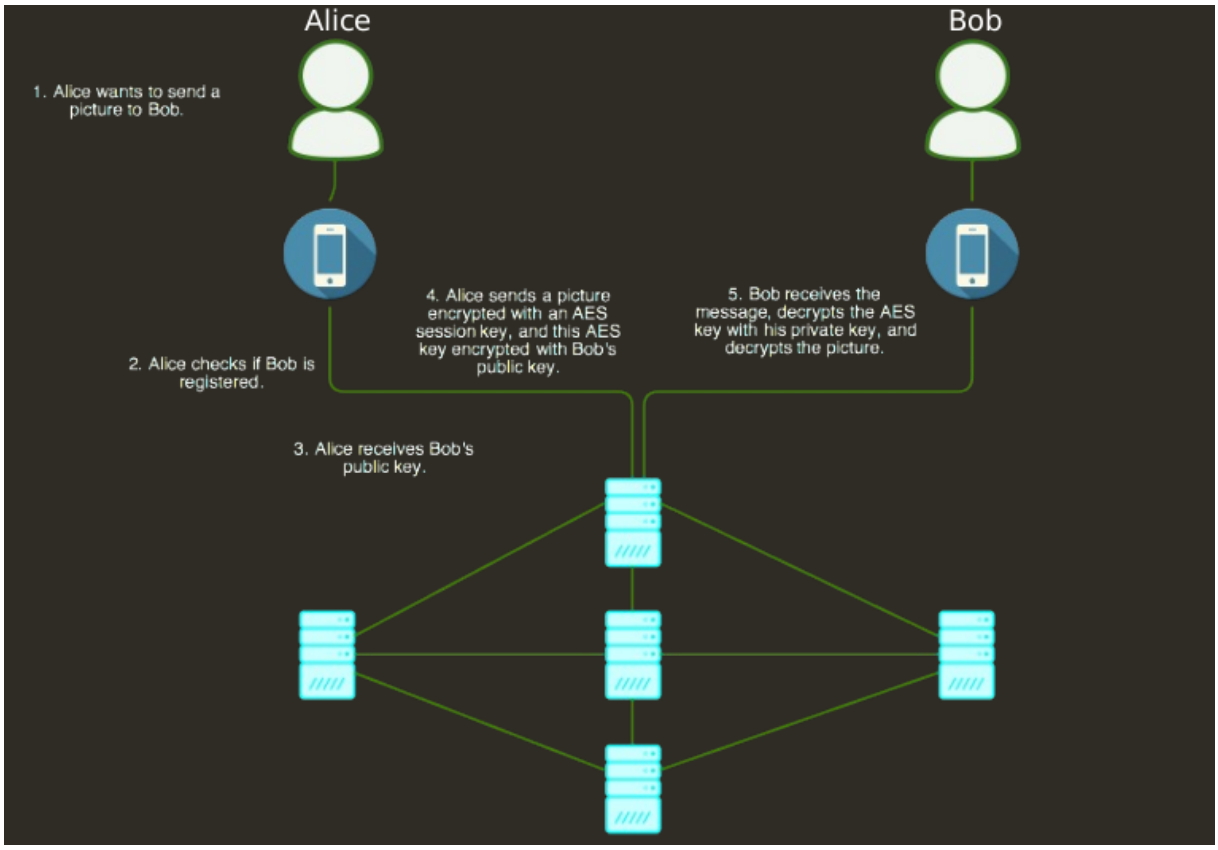

Figure 1: Puzzlr: General Architecture.

3.1 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 then sent to the server to be stored on the database (figure 2).

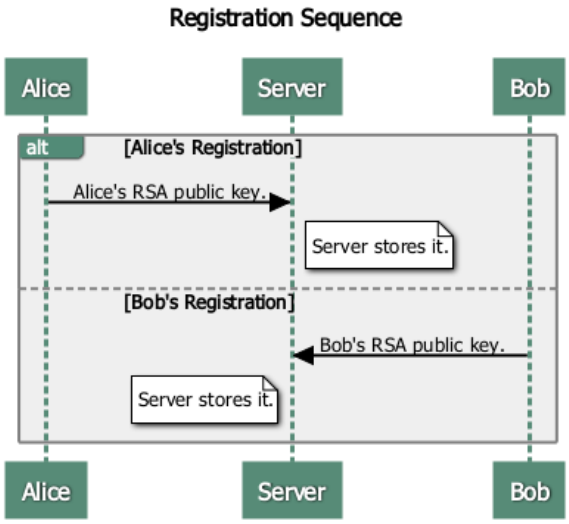

Figure 2: Registration phase Sequence Diagram.

3.2 Retrieval of 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).

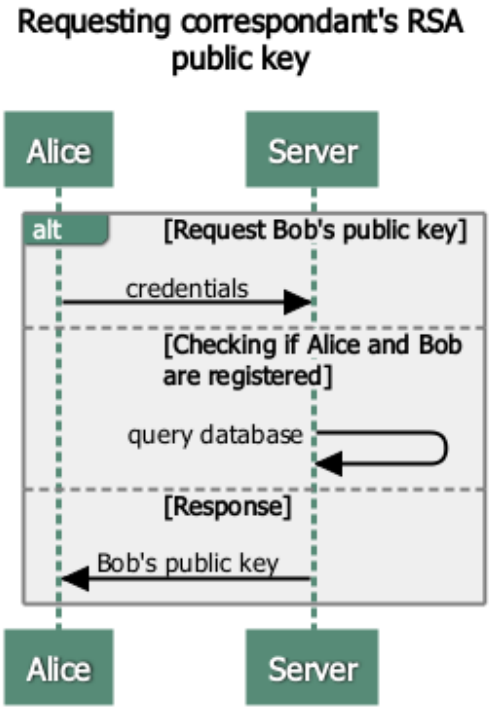

Figure 3: Requesting Correspondant's RSA public key Sequence Diagram.

3.3 Sending and Receiving Picture

After retrieving 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).

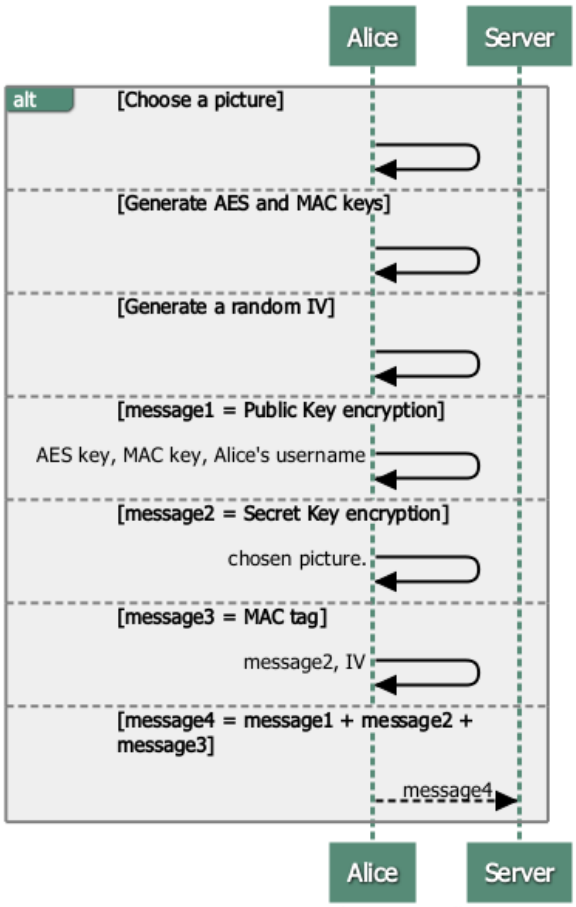

Figure 4: Picture Sending

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.

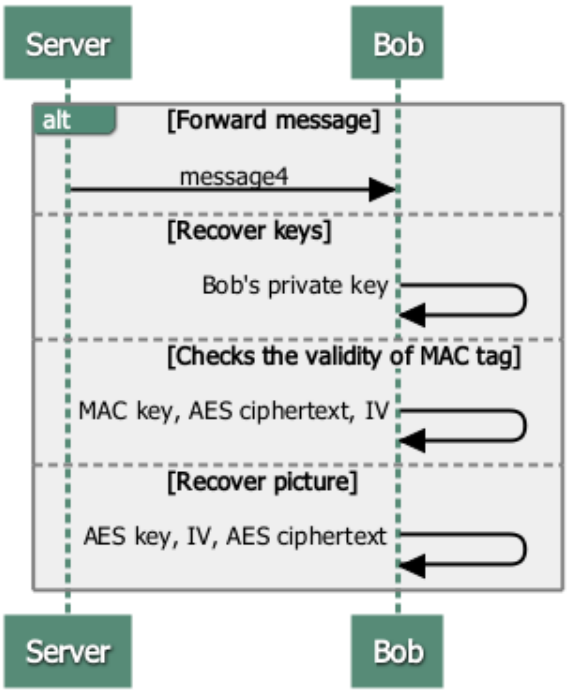

Figure 5: Picture Receival

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 a secure 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.

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.

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

- 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

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 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 ***Hmac-SHA512*** algorithm specification.	256 bits.
tag	Computed only on the AES ciphertext and the IV.	256 bits.

Table 1: Cryptographic primitives.

5 Implementations

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

5.1 Server side

5.1.1 Requirements

For this implementations, we were looking for a decentralized and secure open source server. The server should have some kind of API in order to easily develop with. The idea of a blockchain based server was a natural choice as Bitcoin and others cryptocurrencies can provide a very reliable decentralized ledger.

5.1.2 Fabric

What is Fabric ?

> The Fabric is ledger of digital events, called transactions, shared among different participants, each having a stake in the system. The ledger can only be updated by consensus of the participants, and, once recorded, information can never be altered. Each recorded event is cryptographically verifiable with proof of agreement from the participants.
>
> Transactions are secured, private, and confidential. Each participant registers with proof of identity to the network membership services to gain access to the system. Transactions are issued with derived certificates unlinkable to the individual participant, offering a complete anonymity on the network. Transaction content is encrypted with sophisticated key derivation functions to ensure only intended participants may see the content, protecting the confidentiality of the business transactions.
>
> The fabric is an implementation of blockchain technology, where Bitcoin could be a simple application built on the fabric. It is a modular architecture allowing components to be plug-and-play by implementing this protocol specification. It features powerful container technology to host any main stream language for smart contracts development. Leveraging familiar and proven technologies is the motto of the fabric architecture.

Fabric was released in December 2015[2] and is part of the Linux's Foundation Hyperledger Project[3] which is a "collaborative effort created to advance blockchain technology by identifying and addressing important features for a cross-industry open standard for distributed ledgers that can transform the way business transactions are conducted globally."

[2]See `https://github.com/openblockchain`, now moved at `https://github.com/hyperledger/`
[3]See `https://www.hyperledger.org/`

Fabric is written mostly in Go and a REST API is implemented.

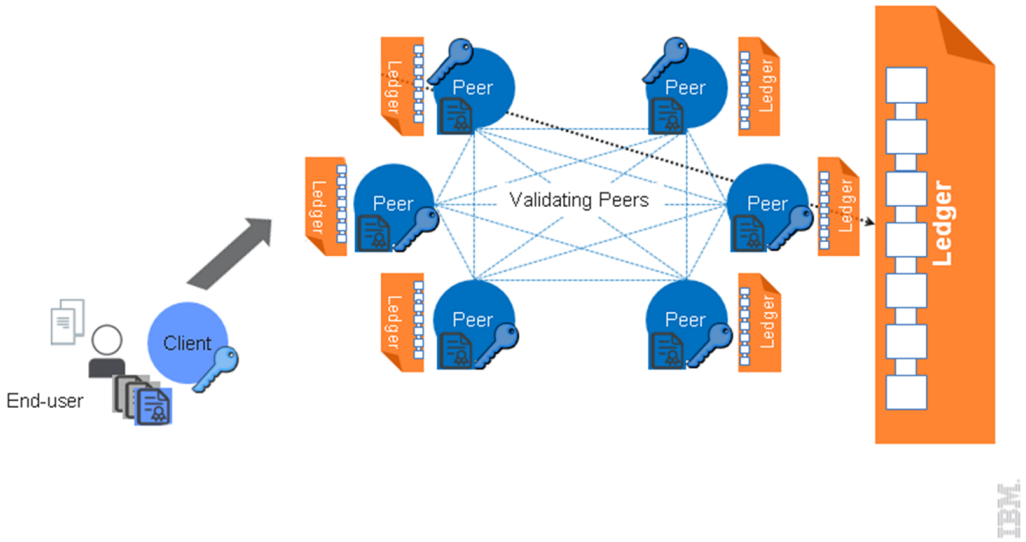

Figure 6: Fabric Overview

Why use Fabric ?

Building our server over the Fabric allowed us to build an unstoppable application. An App that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference.

How we worked with Fabric

For our application to work, we used 3 tables (aka Chaincodes) :

- User, store username and hashed password.
- Key, store the public keys.
- Data, store the data.

This tables are written in Go and available on the Github of the project. We will not review here the code but we can highlight that these chaincodes were developped with security in mind (e.g: the ledger will not reveal if it was the username or the password that was wrong in order to avoid user enumerations).

We released Fabric4J[4] the first Java API for Fabric. There is also a Swift equivalent embedded in the iOS app but we do not plan to release it at this time (as it is less complete and was just created for the purpose of being used in Puzzlr).

[4]Available at `https://github.com/quentinlesceller/Fabric4J`

5.2 Client side

5.2.1 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 Development Environment (IDE). The application is compatible with almost all versions of Android.

5.2.2 IOS

We also wrote an iOS version of Puzzlr. This version was written in Swift and was challenging because Swift is a relatively new language and we had to use a lot of different frameworks and modify them in order to have a working application. The code is made to be readable with a lot of comment and comprehensive variables. We used 6 frameworks and forked 4 of them.

Name	Author	License	Forked	Usage
AsymmetricCryptoManager	Ignacio Nieto Carvajal	MIT	Y	RSA Keys Generation
CryptoSwift	Marcin Krzyanowski	MIT	N	AES and MAC
CryptoImportExportManager	Ignacio Nieto Carvajal	MIT	Y	Key management
RSAUtils	Thanh Nguyen	MIT	Y	RSA Encryption
SwiftyRSA	Los Di Qual	MIT	Y	Key management
SWRevealViewController	Joan Lluch	MIT	N	Side Menu

Table 2: List of frameworks used and forked in Puzzlr iOS

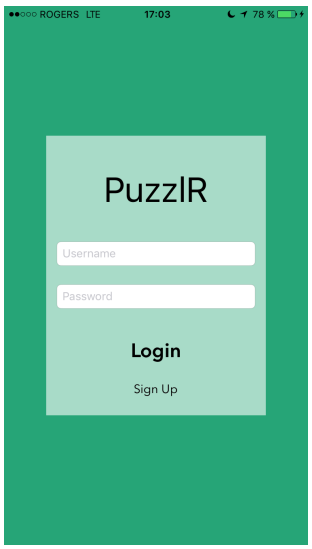

Figure 7: Login page of Puzzlr iOS

When opening the app for the first time, the user is prompted to register. Once he is registered, a keypair is generated and stored securely in the iOS keychain. He can now add contact in the contact menu by writing his friend's username. If the previous friend is registered the app will retrieve his public key and store it in the iOS keychain. The user only have to create a new message, entering his friend's username and adding a photo. His friend will receive the picture instantaneously on his iDevice.

6 Conclusion and Future Work

Puzzlr was a quite challenging and interesting work which helped us to learn a lot of new concepts and to improve our programming skills. Puzzlr offers strong and secure images exchange between different clients, however, there are some missing parts that we did not address yet like the cross-platform compatibility, certificate distribution between nodes, and threat modeling (you can find a draft on a Puzzlr's Microsoft Threat Model in the Github in the "Docs" directory).

For now, the two implementations work only if the two clients use the same platform, in other words, if Alice is using Puzzlr IOS version, she can only communicate with Bob if this one is using the same IOS.

We released our software under MIT License because we believe that the developper community can modify it and enhance it in order to make it more secure and reliable. In the future, once we are sure that Puzzlr offers a decent level of security, we might plan to deploy the backend on different servers around the world and released the apps on their respective store.