Crypto ©

Authors:

- Achraf Feydi
- Mohamed Saïd Fayache

This project is a demonstration of some cryptography algorithms using Flutter Framework including:

- Coding/Decoding.
- Hashing / Hash cracking.
- Symmetric encryption/decryption.
- Asymmetric encryption/decryption.

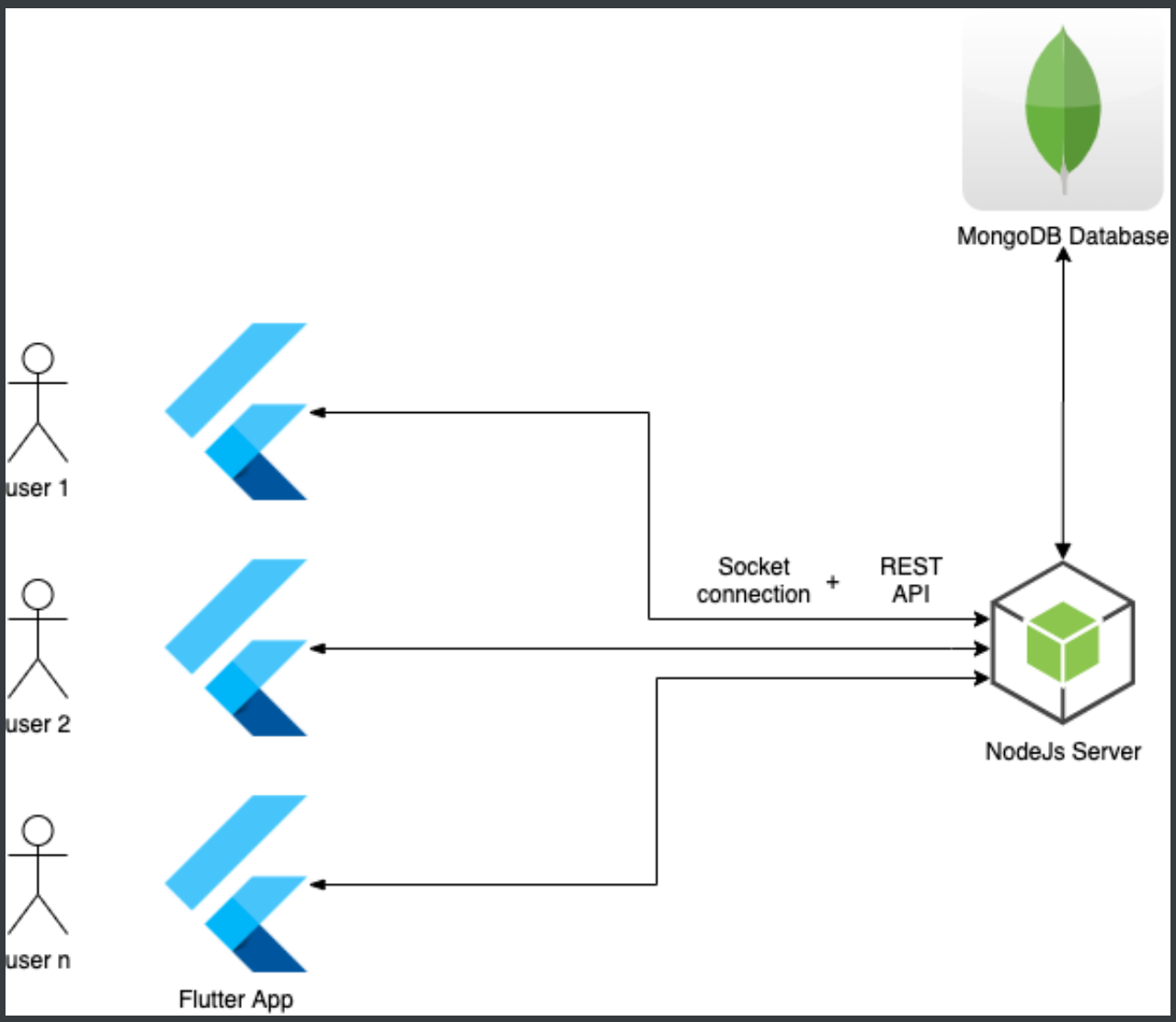
Also for fun, we implemented A **Chatroom App** & **Messenger chat app** inside the main app in order to demonstrate how Symmetric and Asymmetric encyption works .

The message exchange will is performed through **Sockets**.

Architecture

The project is composed of two parts.

- **The Server**: NodeJs App **hosted on heroku** that will play the role of the **bridge** between client and our **Keyserver**.
  - Project Repo: securityProjectServer
- **The Client App**: A Flutter mobile app that contains all our buisness logic.



Demonstration

> “A user interface is like a joke. If you have to explain it, it’s not that good”. — Martin Leblanc

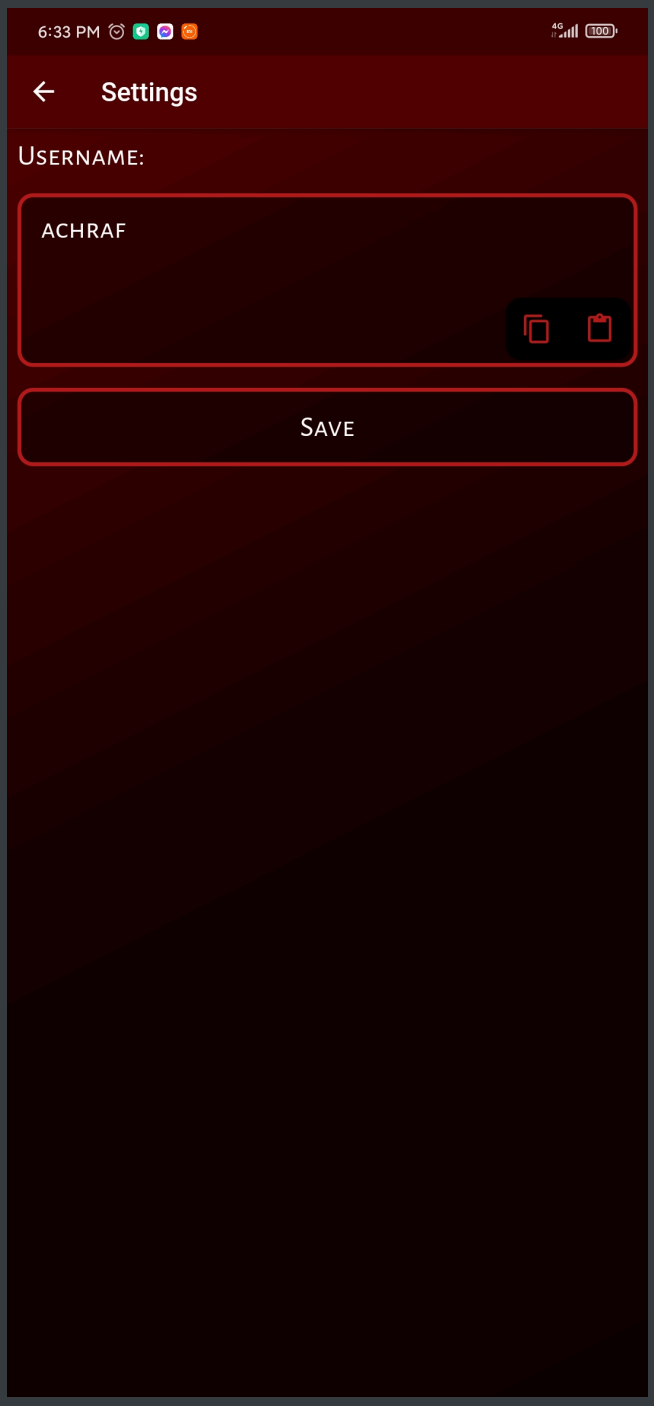
Our User Interface is super **userfriendly**, but for educational purposes 🤦‍♂️ we will explain it.

Now the fun part 😎, Let's discover the app.

Main Screen

Configuration

By clicking the ⚙️ icon on the top of the screen this screen will open up.



From here you can setup the username that will be used as the identifier of our user.

Coding / Decoding

By clicking the `Encoding` button on the Main screen you will get this interface:

Encoding:

Encoding support 3 coding algorithm:

- To Base64
- To Binary
- To Ascii

To illustrate, this is the output of running encoding `Hello world` in :

- Base64: `aGVsbG8gd29ybGQ=`
- Binary: `1101000 1100101 1101100 1101100 1101111 100000 1110111 1101111`

`1110010 1101100 1100100`

- Ascii: `104 101 108 108 111 032 119 111 114 108 100`

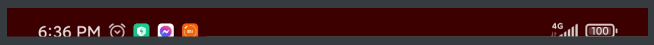
Decoding

Nothing special, It just reverse the operation of encoding.

Hashing

In this part you'll see how we can hash text and crack the hashing using a **brute-force attack** .

Hashing:



You can choose of the following Algorithms for hashing:

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512
- MD5

To illustrate, this is the output of running hashing `Hello world` in :

- SHA-1 : `2aae6c35c94fcfb415dbe95f408b9ce91ee846ed`
- SHA-224 : `2f05477fc24bb4faefd86517156dafdecec45b8ad3cf2522a563582b`
- SHA-256 : `b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9`
- SHA-384 : `fdbd8e75a67f29f701a4e040385e2e23986303ea10239211af907fcbb83578b3e417cb71ce646efd0819dd8c088de1bd`
- SHA-512 : `309ecc489c12d6eb4cc40f50c902f2b4d0ed77ee511a7c7a9bcd3ca86d4cd86f989dd35bc5ff499670da34255b45b0cfd830e81f605dcf7dc5542e93ae9cd76f`
- MD5 : `5eb63bbbe01eeed093cb22bb8f5acdc3`

Cracking the Hash

For the **brute-force attack** we are using a **5 Milions word** dictionary downloaded from this link .

Because of the huge computational power required for this operation we had to divide the dictionary **mini-batchs** with 10000 words in each of them.

Also we executed the code using the `compute` function to do all the work on a different "Thread" to avoid skipping frames and give a feed back on the operation's progress since it takes a long time.

Test result :

Device : Xiaomi Note 7

- Qualcomm SDM660 Snapdragon 660 (14 nm)
- 8gb RAM

Test on the 5 milions words : 59.8525 minutes

Tests per second : 1392 test/s

Symmetric encryption

This is the screen you'll see when you open go to `Symmetric Encryption` from the main screen.

Encrypt

For symmetric encryption we need, in addition to the text, a **key** that both side of the encrypted communication knows.

Any one with the **key** can encrypt and decrypt messages.

You can choose of the following Algorithms for Encryption:

- AES CBC
- AES CFB-64
- AES CTR
- AES ECB
- AES OFB-64/GCTR

- AES OFB-64
- AES SIC

To illustrate, this is the output of running encryption `Hello world` using `VincentRijmen` as the Key :

- AES CBC : `eLVXrWTx1oBdRqy9PAbcAw==`
- AES CFB-64 : `lWc7T08TGmJzU3nP6hS8nQ==`
- AES CTR : `lWc7T08TGmLYvH25MZR7IQ==`
- AES ECB : `eLVXrWTx1oBdRqy9PAbcAw==`
- AES OFB-64/GCTR : `GGkCZJK5inYtcyzxpCVEug==`
- AES OFB-64 : `lWc7T08TGmL8Y1Ip3KiWEg==`
- AES SIC : `lWc7T08TGmLYvH25MZR7IQ==`

Decryption

Nothing special about it, It just reverse the operation of encryption using the same **key**.

ChatRoom 🏠

> The Server is hosted online, you can download and try the app with your friends 👨‍💻

To enter a chatroom the user should specify the **roomName** and the **key** used for symmetric encryption.

Without the key, the user won't be able to see other people messages.

More than 2 persons can join the room. Actually anyone with the **key** can.

You can see in the screen the cool UI of the Chatroom.

By Clicking any chat item you will be redirected to a page with all the details about that message including :

- The Symmetric Encryption **Algorithm**
- The **key**
- The **Encrypted Message** that is transmitted through internet.
- The **Decrypted message** that was decrypted locally in your device using the **key** .

We don't spy on your messages, we are not **WhatsApp** 😭 , be like us 😎 .

Asymmetric encryption

This is the screen you'll see when you click `Asymmetric encryption` on the main screen.

Key Manager

The key manager allow the user to **generate key pairs** (public key, private key) and upload the public key to the **keyServer**.

- Algorithm: `RSA`
- Key size: `2048 bit`

Encrypt

This Interface allows the user to encrypt a message using a **public key**.

- Algorithm: RSA
- Hash: SHA256
- Padding scheme: OAEP (Optimal asymmetric encryption padding)

Decrypt

This Interface allows the user to encrypt a message using a **private key**.

Sign

This Interface allows the user to sign a message using a **private key**.

- Algorithm: RSA
- Hash: SHA256
- Encoding methods: PSS (Provably Secure Encoding Method for Digital Signatures)

Verify Signature

This Interface allows the user to encrypt a message using the combination of **public key** + message.

Messenger 💌

> The Server is hosted online, you can download and try the app with your friends 👱

Now to the **coolest** part.

First, let's understand how things work then we will see the UI.

Let's say `Bob` want to send a message to `Alice` , this is what going to happen:

1. `Bob` download `Alice` 's **public key** from our KeyServer.
2. `Bob` encrypts the message using `Alice` **public key**.
3. `Bob` signs the message using his **private key**
4. `Bob` send the signature and encrypted message to `Alice`
5. `Alice` download `Bob` 's **public key** from our keyServer.
6. `Alice` decrypt the message using her **private key**.
7. `Alice` verify `Bob` 's signature using his **public key**.

Problems we had ⚠️ :

Life is not always a bed of roses, we learned it the hard way that RSA actually has a maximum size of message to encrypt.

To calculate it we can use this table:

Hash	OVERHEAD	RSA 1024	RSA 2048	RSA 3072	RSA 4096
SHA-1	42	86	214	342	470
SHA-224	58	70	198	326	454
SHA-256	66	62	190	318	446
SHA-384	98	30	158	286	414
SHA-512	130	N/A	126	254	382

Since we are using **RSA 2048** with **SHA-256** then we only have 190 bytes as maximum size.

And if we take in concediration that **OAEP** padding takes 42 bytes, then the remaining is `148 bytes` 🤔.

To solve the issue we had to make our message's size a multiple of `148` by adding extra spaces on the top right. then we divide the message in blocks of `148 bytes` , encrypt each of them and send the concatinated result. 😎

We also send the **size of an encrypted block** so the receiver can reverse the operation and get the message back.

This is an example of a transmetter message:


```
{
    "receiverChatID": "Achraf",
    "senderChatID": "Said",
    "content": {
        "message":
"LP1hplB9IczCgoWBYu3yWCP9K7+uZF3RM0iKwSU6L33SVfS1eMDqW6WYWnQs5SdCsiFLriO
IQU0y8pgouUlrODIR32SGSmgQwMaZJLbHCUvRMub3kBnmTb7iygdtHq6kzEWBydAsZ4iIjQ8
jdg3MeJk/pHLaeDRJzM/dv8eg+QXZgcQbsJRk5KfFtnwMkvyzq1lYZA5Q2f+8rsfnbCzJ4OE
sqfXPpsB/MXwrf3m1sxvuM/TeXZvPVeLvJ546pcnKKusDYRoqiGlEM/pfmB2ESle+VC6VyMX
Ck603JSwhTSPXZehVqFnGgNWLnJ20K+VTtV212DGHRjE3jsrNImGsJw==",
        "blockSize":344,
 "signature":"SaQYhD+U6gXPz1Go+otOPiGRPkaiwQZuI4bcDEiuwTvVgivcbF6uE2v37o
aI5tQC9HeusEIou0INGLSLxxO+J91gQ5tKuW3tiUi9L6JNwJ4UHb8j0Ucfjvdon41xqt2duV
nq5p+/67SlvSKBOR/3CIOdFKMuF9XZhCkwsDfj+PkNBCBGfx1kpQTlLEFYWWmf2X8TeTvxn6
Kch3tHORft5gVhpAxBCCPLEN2+5NWe9zJYqhiRJA28Sy15J2wiSpqKNn2MM2HY3biez38lt2
fbnJGfaKJ0fz+NRoxI2czB2TBbQilZRtJ9bW0ngQSmp/DZMnHLy4Xjcq0X61qG/2p06g=="}
}
```


Messenger

By clicking the ➕ button on the bottom of the screen, the user gets the list of users who published their public keys on the keyServer.

By clicking on a user name, our user will be redirected to the chat screen.

By clicking on a received message the user can see all the details of the communication process.

And back to the messenger screen, the user can switch between different conversations.

We are not saving messages locally, so once the user leaves the app, all data will be removed from the RAM.