The Chinese University of Hong Kong
Academic Honesty Declaration Statement

Submission Details

Student Name	WONG Shing (s1155109027)		
Year and Term	2020-2021 Term 2		
Course	IERG-4999-CJ01 Final Year Project II		
Assignment Marker	Professor CHAU Sze Yiu		
Submitted File Name	Submitted Final report.pdf		
Submission Type	Individual		
Assignment Number	4	**Due Date (provided by student)**	2021-04-27
Submission Reference Number	3001175	**Submission Time**	2021-04-26 16:16:56

Agreement and Declaration on Student's Work Submitted to VeriGuide

WONG Shing	2021 April 26
Signature (WONG Shing, s1155109027)	Date

Instruction for Submitting Hard Copy / Soft Copy of the Assignment

IERG4999C Final Year Project II

Encrypted Cloud Storage for Smartphones

BY

WONG SHING
1155109027

A FINAL YEAR PROJECT REPORT
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR OF INFORMATION ENGINEERING
DEPARTMENT OF INFORMATION ENGINEERING
THE CHINESE UNIVERSITY OF HONG KONG

April 2021

Abstract

There are many existing cloud storage encryption apps in the market like Boxcryptor and Cryptomator. However, they have their disadvantages. In this paper, we propose a third-party cloud storage encryption app that learns from their advantages and overcomes their weaknesses. We use ChaCha20-Poly1305 and ECC respectively for symmetric and asymmetric key encryption while ECDH for ECC key exchange. These encryption algorithms are mature enough to be put on the market and are much more efficient and secure than those used by other apps. Their detailed difference can be found in our evaluation part.

1. Introduction

Cloud storage has been widely used by the public nowadays, especially for Google Drive, which is one of the famous cloud storage platforms. However, one year ago, Google made a technical mistake [1]. He mixed up the way of separating individual clients' accounts and cloud storage, resulting in sending one's Google Photos to unrelated users [1]. Though he claimed that less than 0.01% users were affected, this incident did show that cloud service providers cannot be 100% trusted.

Besides, for file sharing using Drive, there have been over 4000 incidents reported last year [2]. The incidents included "loss of inadequately protected electronic equipment" and "unauthorized disclosure of personal data" [2]. From the titles, we can conclude that encryption of data during sharing and data backup are becoming necessary to prevent personal data leakage.

This paper introduces a third-party app that allows cloud files encryption, cloud files sharing authentication and cloud files backup. They are achieved by: Encrypting cloud data with ECC (Elliptic-curve cryptography) algorithm, Using ECDH (Elliptic Curve Diffie-Hellman key exchange) for key exchange and Using Dropbox for back-up.

2. Related Work

Cryptomator [3] is a third-party app that allows cloud files encryption. It uses AES (Advanced Encryption Standard) CTR (Counter) mode as the encryption algorithm. Random numbers and salt are added during the encryption process to enhance the security. HMAC (Keyed-Hashing for Message Authentication) SHA-256 (Secure Hash Algorithm) is used for integrity check. However, Cryptomator does not support files sharing authentication, which means it is only for private use.

CTR (Counter) mode is regarded as the mode in AES that provides top security performance. However, it still has its weaknesses. First, comparing to AES GCM (Galois/Counter) mode [4] and ChaCha20-Poly1305 [5], it does not provide any integrity check. HMAC or GMAC (Galois Message Authentication Code) are needed as well for message authentication. Second, the security level of CTR mode entirely depends on the counter, that the counter value must not be reused [6]. Otherwise, all the security is lost. CTR behaves like a single-point failure, that one malfunction or mistake will lead the whole encryption useless [7].

3. Methodology

3.1 Overview

Fig. 1 shows the app layout. There will be five buttons with five functions naming: encrypt, decrypt, add receiver, my public key and Dropbox.

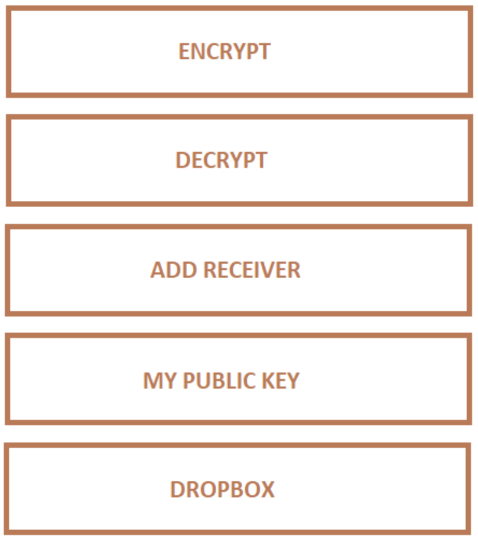


Fig. 1 App Layout

Fig. 2 shows the process of encryption. When the 'encrypt' button is pressed, a dialog is shown with two choices: encrypt for own use and encrypt for sharing. After that, different actions are taken, including selecting files from local file system and google drive, inputting a password and selecting the receiver's public key.

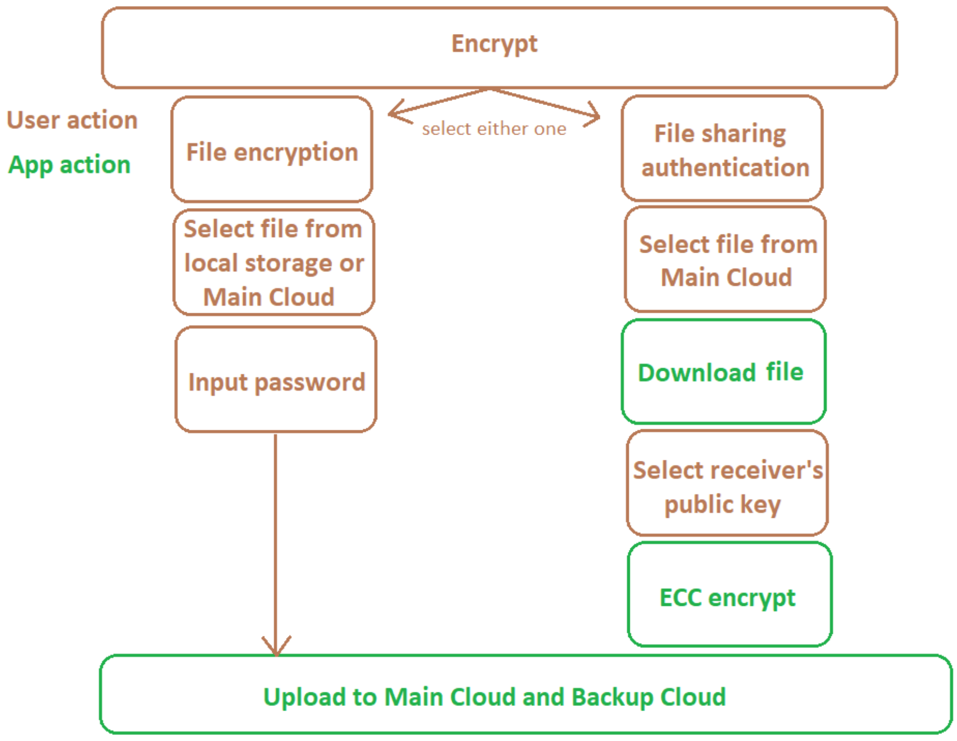

Encrypt
User action
App action
File encryption
select either one
File sharing authentication
Select file from local storage or Main Cloud
Select file from Main Cloud
Input password
Download file
Select receiver's public key
ECC encrypt
Upload to Main Cloud and Backup Cloud

Fig. 2 Encryption Flowchart

Fig. 3 shows the process for decryption. When the ‘decrypt’ button is pressed, a dialog with two choices: own files and shared files is displayed. Then, the user selects the file to download from Google Drive. After that, different actions are taken. For shared file, the user is required to select the sender’s public key before inputting the password.

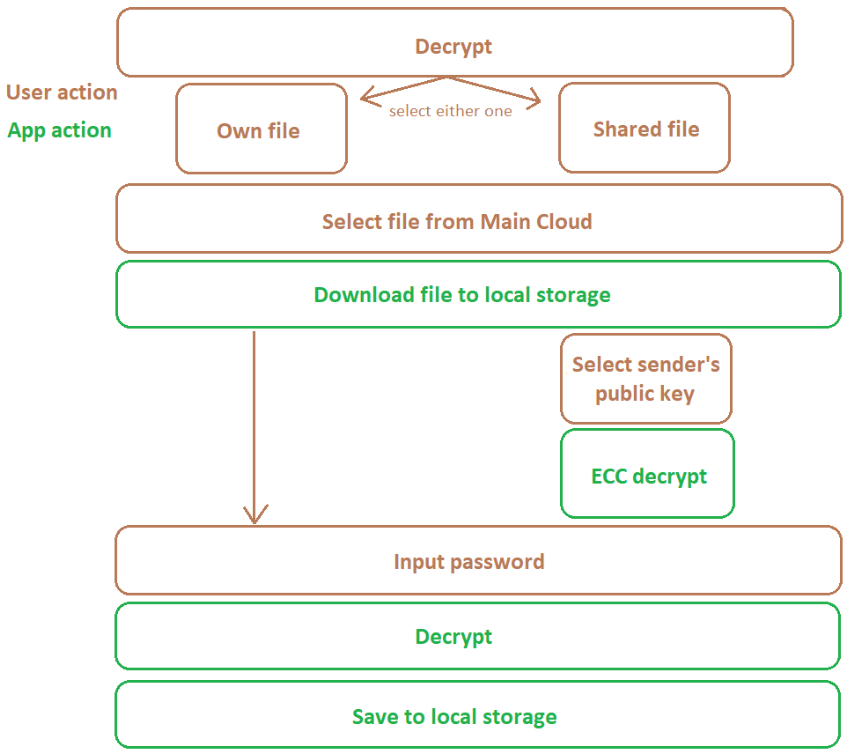


Fig. 3 Decryption Flowchart

Fig. 4 shows the steps for adding a receiver. When the ‘add receiver’ button is clicked, the user can either scan the QR code of the receiver’s public key or import the QR code from the gallery. Then, the user is required to set a name for the receiver. After that, the receiver’s public key is saved in a folder for future use.

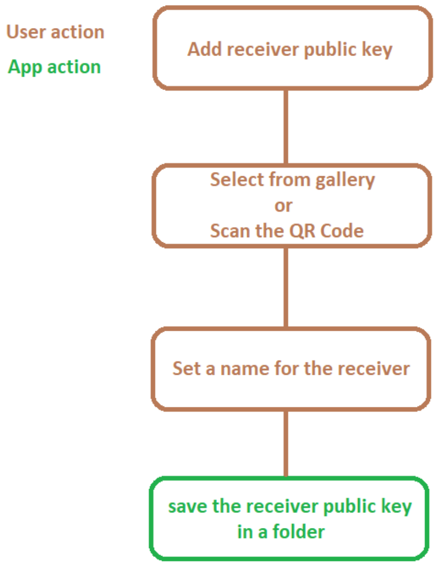


Fig. 4 Add Receiver

When the ‘my public key’ button is clicked, a page that looks like Fig. 5 is shown. At the top of the page, the user can find the file path to his own public key text file. Below of it, it will a QR code that stores his public key string. He can choose to send either the text file or the QR code to the receiver.

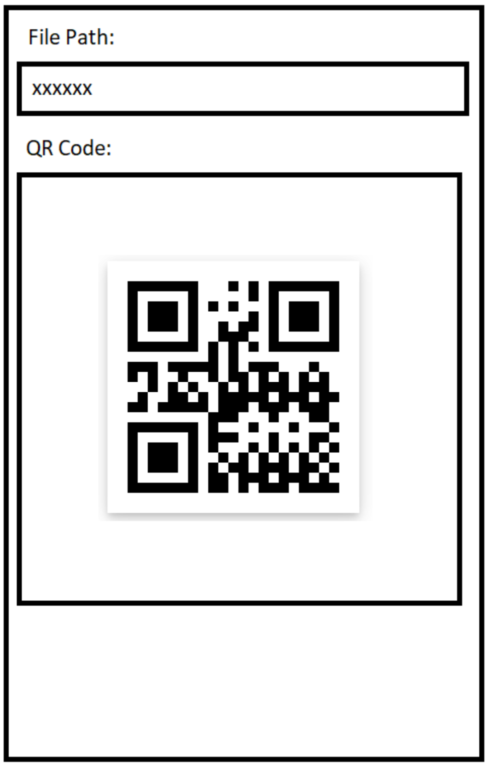


Fig. 5 My Public Key

Fig. 6 shows the steps for logging in the backup cloud storage, which is Dropbox. When the ‘Dropbox’ button is clicked, the user is directed to the Dropbox login page. If the login in succeeds, the access token will be automatically saved in the private memory of the app for future authentication.

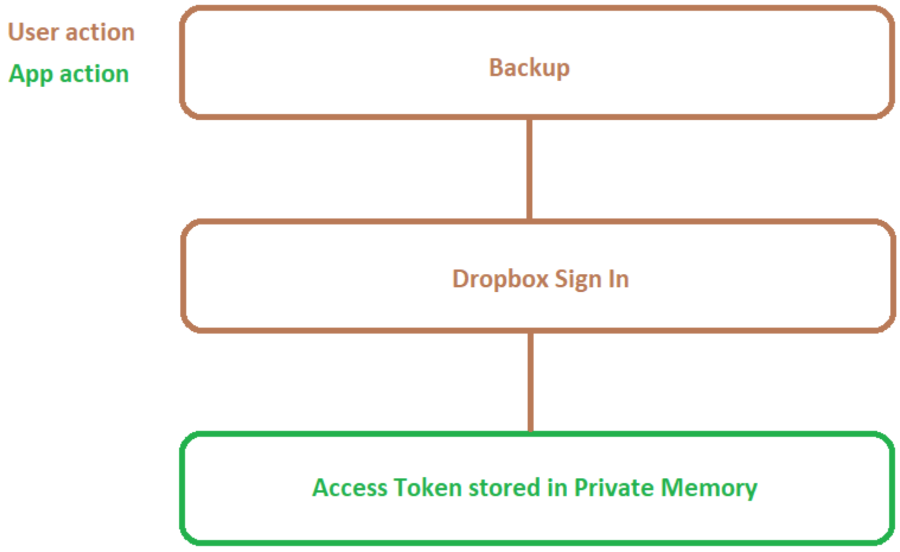


Fig. 6 Dropbox Backup

3.2 UI (User Interface)

We have improved the UI to provide a better user experience to our clients.

First, the input text will be set to ‘password’ type, that means while the client is inputting the password, the plaintext is replaced by ‘*’. Fig. 7 shows the simulation of the UI.

Fig. 7 Hidden Password

Second, we encapsulate our functions and provide a clear layout. Only five buttons are displayed: ENCRYPT, DECRYPT, ADD RECEIVER, MY PUBLIC KEY, DROPBOX. Fig. 1 shows the app layout. After collecting the necessary input from the client, the app will do the remaining job itself without the need of user’s interruption. In Fig. 2, Fig. 3, Fig. 4 and Fig. 6, the brown boxes are user’s interruption while the green boxes are the background process.

3.3 ECC (Elliptic-curve cryptography)

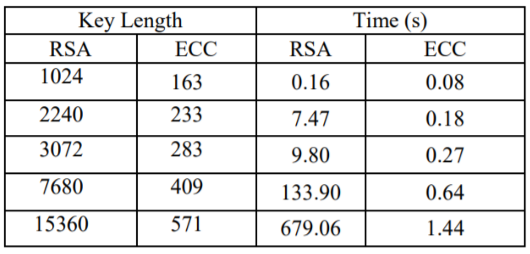
There are two common asymmetric encryption algorithms in the market which are ECC and RSA. For our app, we choose ECC as our public key encryption algorithm.

<u>Reason to choose ECC instead of RSA:</u>

We have considered their security performance, speed, and practicability.

In terms of security performance, there are research papers stating that ECC is more secure than RSA [8], [9], [10], [11], especially for resource constraint devices like smartphones [9], because ECC can attain a comparable security level to RSA with a shorter key size [11]. Besides, ECC is concluded to have a better security performance on software security, hardware security and Wireless LAN security [8]. According to the above research papers, it can be concluded that ECC performs better than RSA from the perspective of security.

In terms of speed, ECC requires a shorter time to generate the key pair because it has a shorter key length [9], [11]. Fig. 8 shows that the time needed for ECC 163 bits key generation is only half of that for RSA 1024 bits. Moreover, increase of the key length will make the difference of the time required to becomes more obvious. Therefore, it can be concluded that ECC performs better than RSA in the aspect of speed.



Key Length		Time (s)	
RSA	ECC	RSA	ECC
1024	163	0.16	0.08
2240	233	7.47	0.18
3072	283	9.80	0.27
7680	409	133.90	0.64
15360	571	679.06	1.44

Fig. 8 Key Generation Performance of RSA and ECC [10]

In terms of practicability, we found that Google has started to use ECC for key generation from 2011 [12], which shows that ECC is mature enough to be put in the market.

After considering the above three factors, we pick ECC as our asymmetric encryption algorithm due to its safety, speed, and maturity.

ECDH (Elliptic Curve Diffie-Hellman key exchange):

The key exchange principle can be described as the graph below:

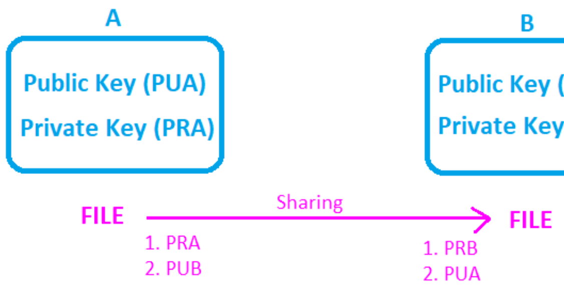


Fig. 9 Key Exchange

PRA and PUA represents the private key of A and public key of A.

PRB and PUB represents the private key of B and public key of B.

For one asymmetric key pair, only the private key can decrypt the public key. For instance, from Fig. 9, PRA can decrypt PUA while PRB can decrypt PUB. Moreover, the public key can be known by others while the private key must be kept secretly. For instance, from Fig. 9, PRA and PRB should be kept secretly while PUA and PUB can be shared to others.

Fig. 9 simulates the key exchange process. Assume A is the sender and B is the receiver. Assume they have generated their own pair of asymmetric keys and shared their public key to each other. When A sends a file to B, he first encrypts the file with PRA and PUB. When B receives the file, he decrypts the file with PUA and PRB.

Device Authentication

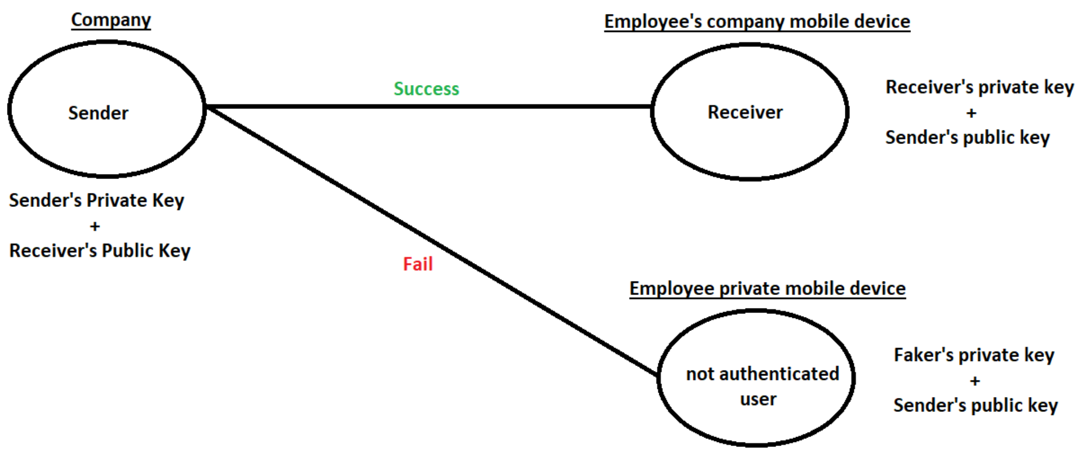


Fig. 10 Device Authentication Flowchart

Each device has only one key pair consisting of one public key and one private key. The public key can be found in the external storage of the mobile device while the private key is kept secretly. Our app is said to be device authenticated because no other mobile device can get the same private key. In the business scenario, only the authenticated mobile device can successfully decrypt the shared company file. This helps to prevent the leak of trade secret.

Fig. 10 simulates a process for device authentication. The company acts as a sender sends a file to a targeted mobile device, which is the employee's company mobile device. The company first encrypts the file with his own private key and the public key of the targeted mobile device. When the employee receives the file, he can only decrypt the file on the targeted mobile device instead of his private mobile device because the private key of the latter cannot decrypt the public of the former.

3.4 Backup

Reasons of choosing Dropbox as our cloud storage:

Google Drive is the most popular cloud storage over the world, occupying about 34.49% of the market shares [13]. However, it is already used as the main cloud. To reduce the risks, we choose another cloud, Dropbox, which comes after Google Drive with about 20.91% of the market shares [13], to be the backup cloud. Besides, Dropbox performs comparably well with Google Drive in the security aspect. Both support two-factor authentication and encrypt the data at rest as well as the data in transit [14], [15]. For Dropbox, it provides a stronger protection than Google Drive on the data at rest. The former uses AES 256-bit encryption while the latter uses AES 128-bit encryption [15]. This makes Dropbox a better choice to be the backup cloud which serves data at rest most of the time.

Importance of Backup Cloud:

We implement a backup cloud to deal with any accidents happened to the main cloud. If the main cloud is compromised by the hackers or misconfigured some settings, the cloud files may disappear and never be found. Therefore, we add one more cloud as the backup cloud in our app to further protect the user's data.

4. Results

4.1 App layout

Fig. 11 shows the app layout. There are five buttons with five functions.

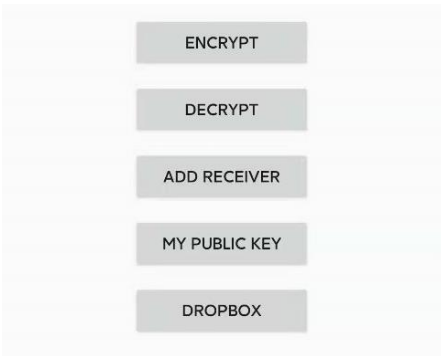


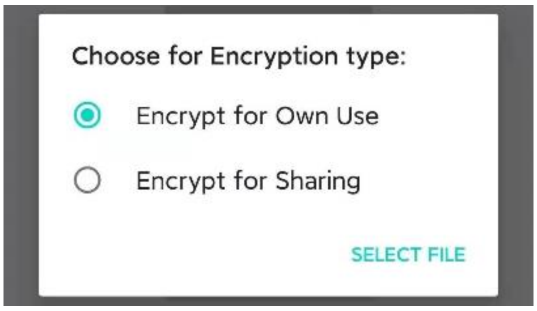
Fig. 11 The App Layout

4.2 Encryption

A dialog is shown after the ‘encrypt’ button is clicked.

Encrypt for Own Use

If ‘encrypt for own use’ is chosen, followed actions are file selection, file name and type set as well as password input.



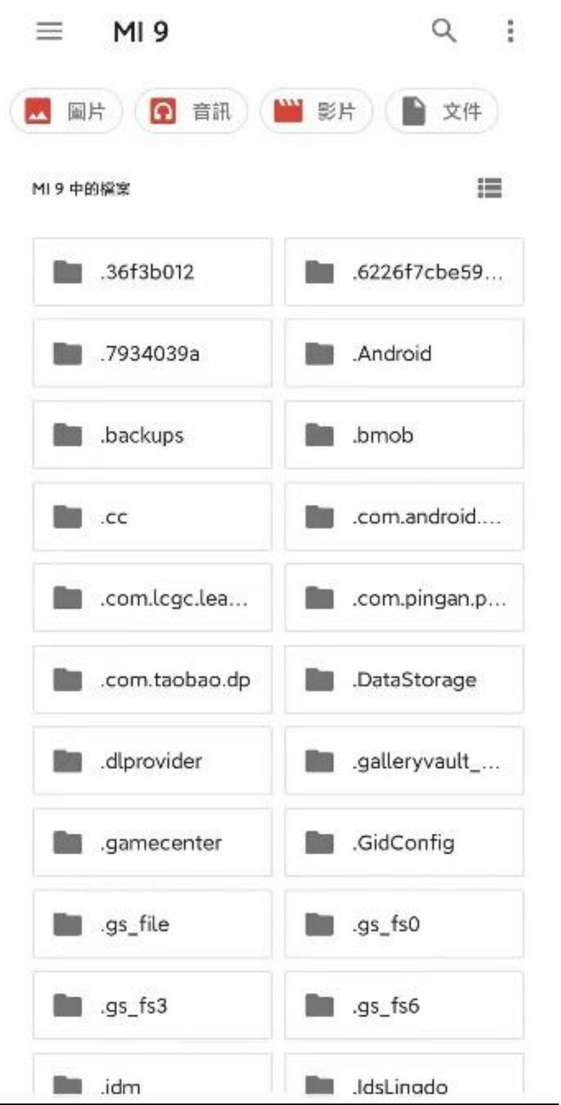
Choose for Encryption type:

[✓ Encrypt for Own Use]

Encrypt for Sharing

SELECT FILE

Fig. 12 Encryption for Own Use Step 1 - dialog



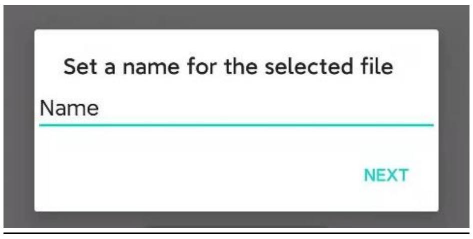
MI 9

圖片 音訊 影片 文件

MI 9 中的檔案

.36f3b012	.6226f7cbe59…
.7934039a	.Android
.backups	.bmob
.cc	.com.android.…
.com.lcgc.lea…	.com.pingan.p…
.com.taobao.dp	.DataStorage
.dlprovider	.galleryvault_…
.gamecenter	.GidConfig
.gs_file	.gs_fs0
.gs_fs3	.gs_fs6
.idm	.IdsLingdo

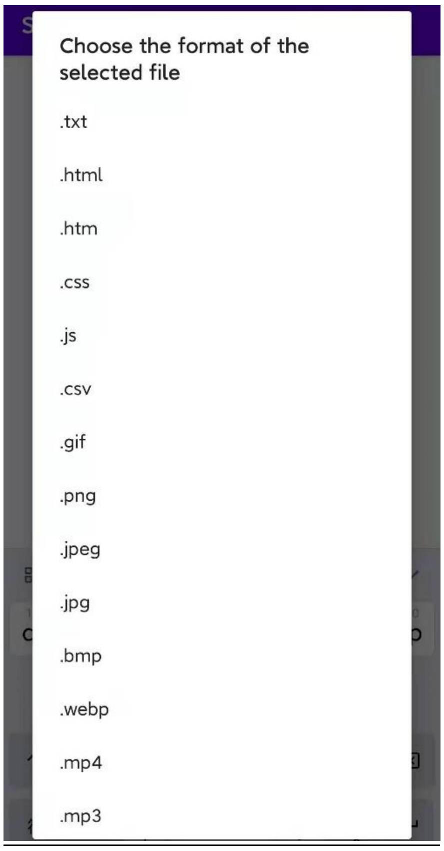
Fig. 13 Encryption for Own Use Step 2 – file selection


Set a name for the selected file

Name

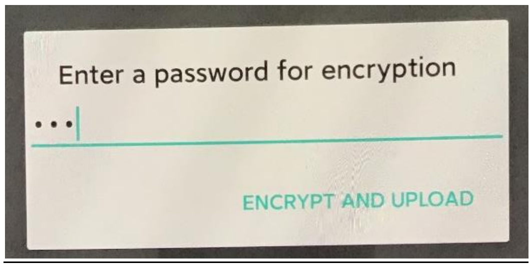
NEXT

Fig. 14 Encryption for Own use Step 3 – file name set


Choose the format of the selected file

.txt
.html
.htm
.css
.js
.csv
.gif
.png
.jpeg
.jpg
.bmp
.webp
.mp4
.mp3

Fig. 15 Encryption for Own Use Step 4 – file type set


Enter a password for encryption

•••

ENCRYPT AND UPLOAD

Fig. 16 Encryption for Own Use Step 5 – password input

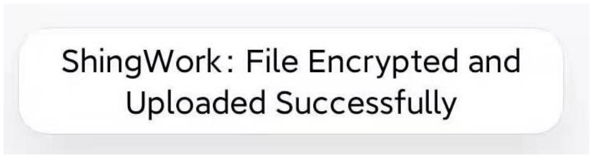
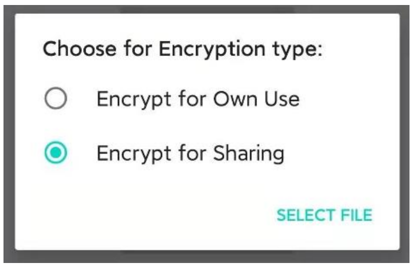

ShingWork: File Encrypted and Uploaded Successfully

Fig. 17 Encryption for Own Use Step 6 – show success/fail message

Encrypt for Sharing

If ‘encrypt for sharing’ is chosen, followed actions are file selection as well as receiver public key selection.


Choose for Encryption type:

Encrypt for Own Use [✓ Encrypt for Sharing]

Fig. 18 Encryption for Sharing Step 1 - dialog

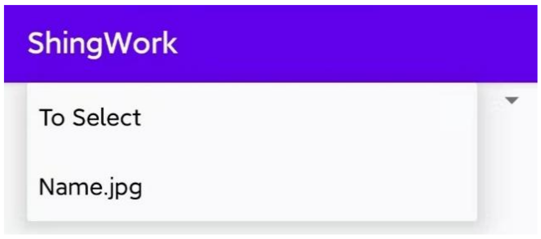
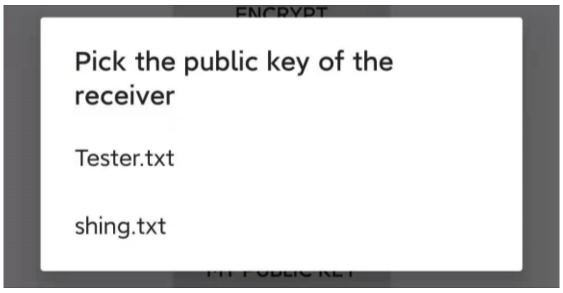


Fig. 19 Encryption for Sharing Step 2 – file selection


Pick the public key of the receiver

Tester.txt

shing.txt

Fig. 20 Encryption for Own Use Step 3 – receiver public key selection

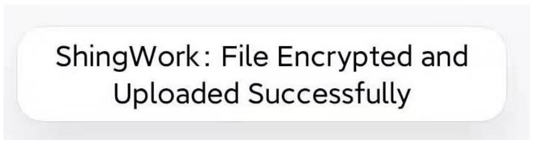

ShingWork: File Encrypted and Uploaded Successfully

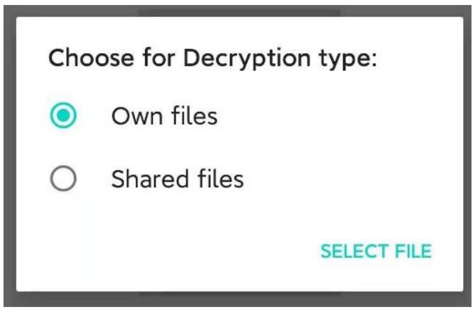
Fig. 21 Encryption for Own Use Step 4 – show success/fail message

4.3 Decryption

A dialog is shown after the ‘decrypt button is clicked.

Decrypt for Own Files

If ‘decrypt for own files’ is chosen, followed actions are file selection and password input.



Choose for Decryption type:

[✓ Own files]

Shared files

Fig. 22 Decryption for Own Files Step 1 - dialog

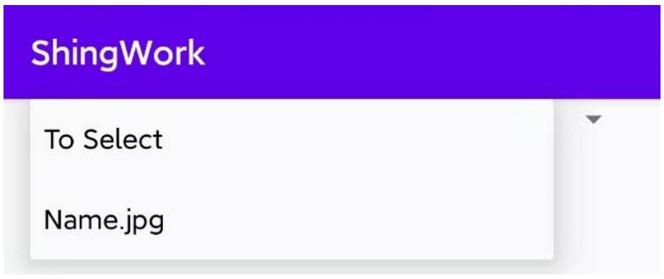
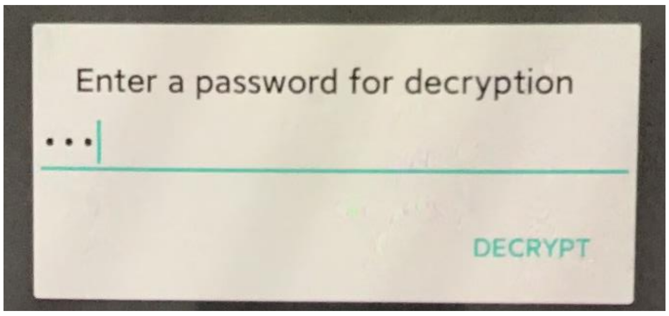


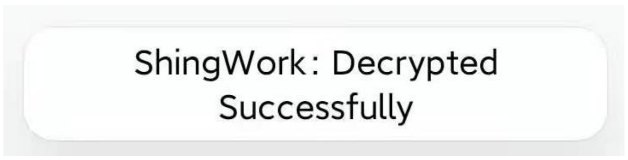
Fig. 23 Decryption for Own Files Step 2 – file selection



Enter a password for decryption

•••

Fig. 24 Decryption for Own Files Step 3 – password input

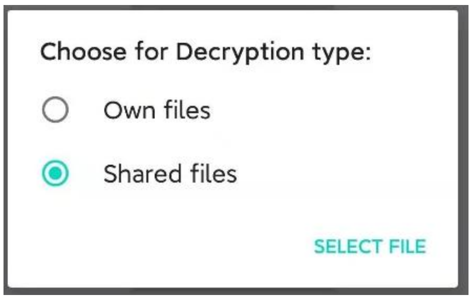


ShingWork: Decrypted Successfully

Fig. 25 Decryption for Own Files Step 4 – show success/fail message

Decrypt for shared files

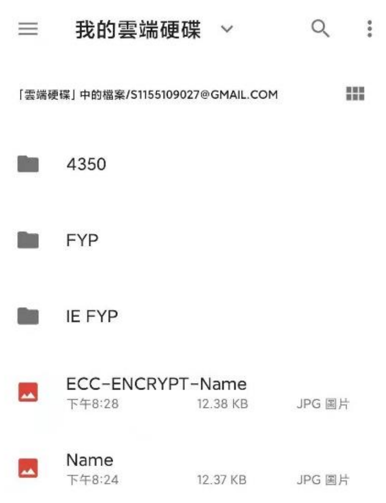
If ‘decrypt for shared files’ is chosen, followed actions are file selection, file name and type set, public key selection and password input.



Choose for Decryption type:

Own files [✓ Shared files]

Fig. 26 Decryption for Shared Files Step 1 - dialog



「雲端硬碟」中的檔案/S1155109027@GMAIL.COM

4350

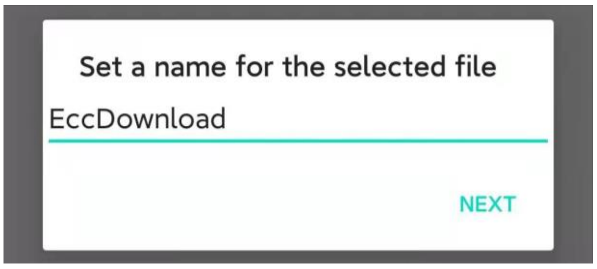
FYP

IE FYP

ECC-ENCRYPT-Name 下午8:28 12.38 KB JPG 圖片

Name 下午8:24 12.37 KB JPG 圖片

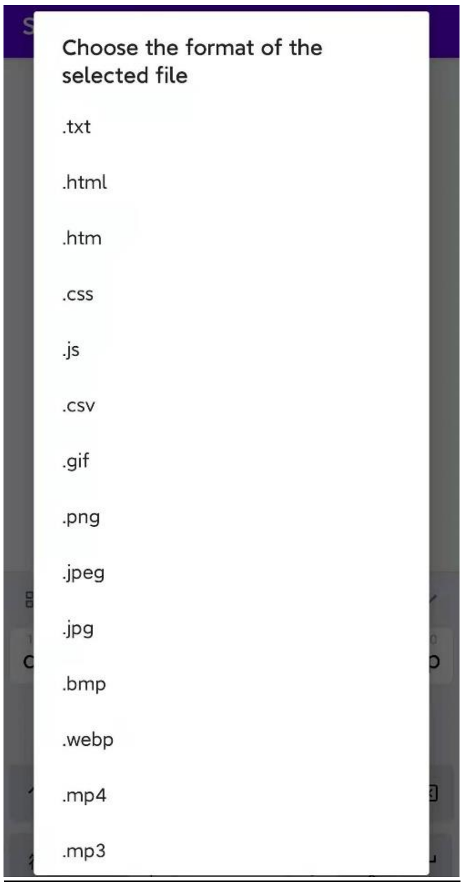
Fig. 27 Decryption for Shared Files Step 2 – file selection



Set a name for the selected file

EccDownload

Fig. 28 Decryption for Shared Files Step 3 – file name set


Choose the format of the selected file

.txt

.html

.htm

.css

.js

.csv

.gif

.png

.jpeg

.jpg

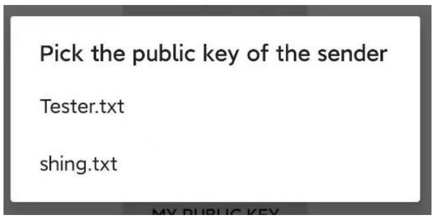
.bmp

.webp

.mp4

.mp3

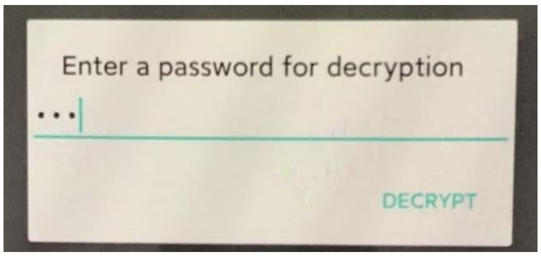
Fig. 29 Decryption for Shared Files Step 4 – file type set


Pick the public key of the sender

Tester.txt

shing.txt

Fig. 30 Decryption for Shared Files Step 5 – sender public key selection


Enter a password for decryption

•••

DECRYPT

Fig. 31 Decryption for Shared Files Step 6 – password input

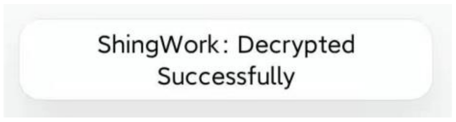

ShingWork: Decrypted Successfully

Fig. 32 Decryption for Shared Files Step 7 – show success/fail message

4.4 Add receiver

Fig. 33 shows the layout of the page after button ‘add receiver’ is pressed. The user can either scan the QR Code or import the QR code from the gallery. After that, the user is required to set a name for the receiver for future use.

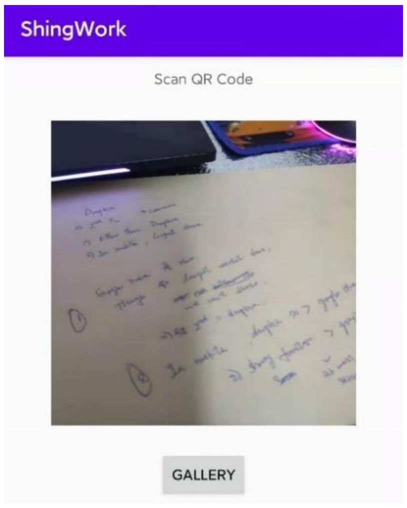
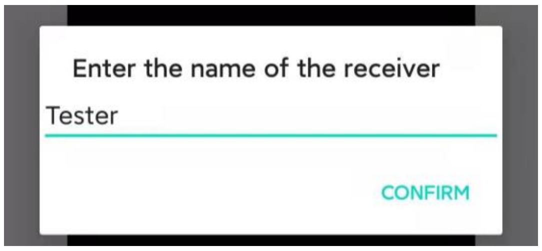


Fig. 33 Add Receiver – Layout

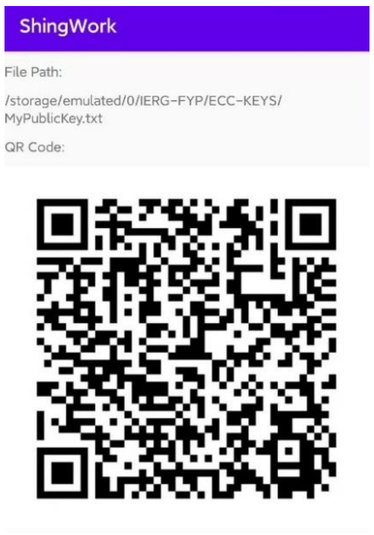

Enter the name of the receiver

Tester

Fig. 34 Add Receiver – receiver name set

4.5 My public key

Fig. 35 shows the layout of the page after button ‘my public key’ is pressed. The file path to the public key text file is shown at the top of the page, followed by a QR Code.



File Path:

/storage/emulated/0/IERG-FYP/ECC-KEYS/MyPublicKey.txt

QR Code:

Fig. 35 My Public Key

4.6 Dropbox

When the button ‘Dropbox’ is clicked, a Dropbox login page is displayed. After successful login, an access token is shown on the screen. The user is required to copy the access token to the box below for two-factor authentication. Also, the access token will be saved in the private memory of the app for future use.

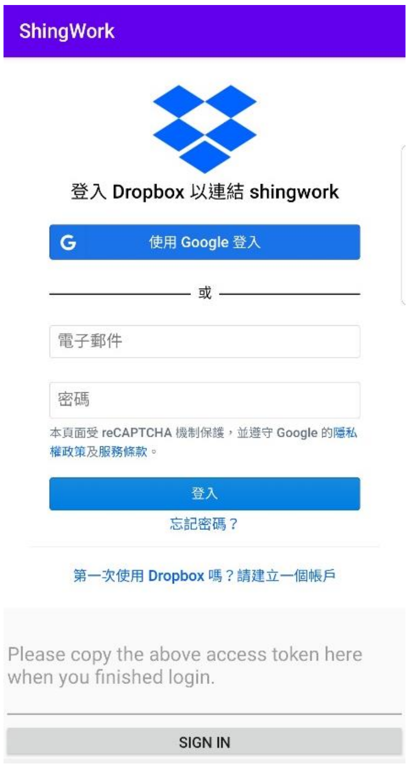
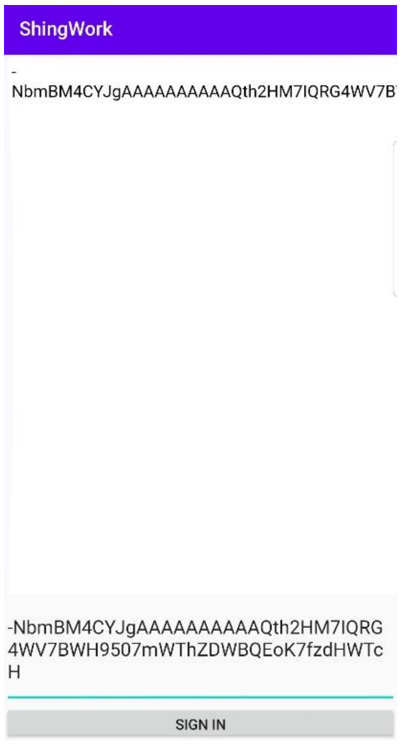

ShingWork
登入 Dropbox 以連結 shingwork
使用 Google 登入
或
電子郵件
密碼
本頁面受 reCAPTCHA 機制保護，並遵守 Google 的隱私權政策及服務條款。
登入
忘記密碼？
第一次使用 Dropbox 嗎？請建立一個帳戶
Please copy the above access token here when you finished login.
SIGN IN

Fig. 36 Dropbox Login Page



-
NbmBM4CYJgAAAAAAAAAAQth2HM7IQRG4WV7B

-NbmBM4CYJgAAAAAAAAAAQth2HM7IQRG
4WV7BWH9507mWThZDWBQEoK7fzdHWTc
H

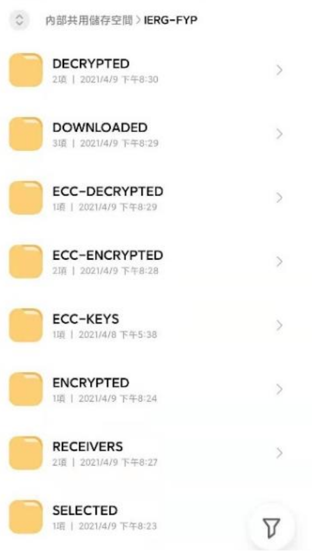
Fig. 37 Client’s access token

4.7 Folder

Fig. 38 and Fig. 39 show the generated folders. Folder 'IERG-FYP' is created once the app is downloaded and started. For every encryption and decryption request, the selected file will be put into corresponding folders, as shown in Fig. 39.

IERG-FYP
8項 | 2021/4/9 下午8:29

Fig. 38 Root Folder



內部共用儲存空間 > IERG-FYP

DECRYPTED
2項 | 2021/4/9 下午8:30

DOWNLOADED
3項 | 2021/4/9 下午8:29

ECC-DECRYPTED
1項 | 2021/4/9 下午8:29

ECC-ENCRYPTED
2項 | 2021/4/9 下午8:28

ECC-KEYS
1項 | 2021/4/8 下午5:38

ENCRYPTED
1項 | 2021/4/9 下午8:24

RECEIVERS
2項 | 2021/4/9 下午8:27

SELECTED
1項 | 2021/4/9 下午8:23

Fig. 39 Generated Folders

5. Evaluation

5.1 Test settings

A lot of research states that ChaCha20-Poly1305 and ECC ECDH respectively performs better than AES and RSA in the encryption time performance. Therefore, we do the evaluation of the algorithms to check their reliability.

There will be two sets of comparison, including the symmetric key algorithms and asymmetric key algorithms. Table 1 and Table 2 respectively shows the comparison details of symmetric and asymmetric key algorithms.

Table 1. Symmetric Key Comparison

Symmetric Key Algorithm	ChaCha20-Poly1305	AES-GCM
Key Spec	PBKDF2withHmacSHA1	PBKDF2withHmacSHA1
Password-based key iteration	1000	1000
Padding	No Padding	No Padding
Key length	256	256
Nonce (bytes)	12	16

For the sake of fairness, they have the same settings. Both are having 1000 times password-based key iteration, no padding as well as key length of 256 bits, and using PBKDF2 with HmacSHA1 as the Key Spec. The nonce bytes are contributed to their own algorithm design, so the difference of the nonce bytes is acceptable.

Table 2. Asymmetric Key Comparison

Asymmetric Key Algorithm	ECDH	RSA
Key Spec	ECC	RSA
Key Length	256	3072
Padding	No Padding	OAEP
Encryption Algorithm	ChaCha20-Poly1305	AES-GCM

We configure the test settings for asymmetric key algorithms according to the common usage on the market. ECDH is usually worked with ChaCha20-Poly1305 while ESA is usually worked with AES-GCM.

The mobile device used in the test is Samsung S8+, with 64-bit octa-core CPU and 6GB RAM (Random Access Memory). The start time is set right before the key generation and the end time is set right after the file encryption (without outputting the file). The time is calculated using System.nanoTime() from Java library, with accuracy to nano seconds.

To simulate different file types, four file sizes are tested, including 102.4KB, 1MB and 10MB, representing of normal text files, images, and short videos. Besides, 100MB is chosen as stress test.

5.2 Test results

Table 3. Encryption time performance for ChaCha20-Poly1305 and AES-GCM with PBKDF2

Algorithm / file size	102.4KB (ms)	1MB (ms)	10MB (ms)	100MB (ms)
ChaCha20-Poly1305 with PBKDF2	27.67	41.33	102.67	644
AES-GCM with PBKDF2	36.67	140.67	1090.67	10611.67
Ratio of AES to ChaCha20	1.33	3.40	10.62	16.48

* time is the average time of three trials

* See Appendix for details

Table 4. Encryption time performance for ECC ECDH and RSA OAEP

Algorithm / file size	102.4KB (ms)	1MB (ms)	10MB (ms)	100MB (ms)
ECC ECDH + ChaCha20-Poly1305 256 bits	3.67	9.33	63.67	550
RSA OAEP + AES-256 GCM 3072 bits	1594.00	1979.67	3003.67	12108
Ratio of RSA to ECC	434.3	212.18	47.18	22.01

* time is the average time of three trials

* See Appendix for details

Table 5. Encryption time performance for ChaCha20-Poly1305 without PBKDF2

Algorithm / file size	102.4KB (ms)	1MB (ms)	10MB (ms)	100MB (ms)
ChaCh20-Poly1305	1.27	5.85	47.56	417.87

* time is the average time of three trials

* See Appendix for details

5.3 Discussion

Table 3 shows the test results of the symmetric key algorithms, ChaCha20-Poly1305 and AES-GCM. PBKDF2 is included in the test because it is part of the encryption, converting the user’s password to symmetric secret key. Same setting of PBKDF2 is applied to the two algorithms. We can see that the former performs better than the latter, with a result of 27.67ms to 36.67ms, 41.33ms to 140.67ms, 102.67ms to 1090.67ms and 644ms to 10611.67ms for files of 102.4KB, 1MB, 10MB and 100MB. Besides, we can see that the time ratio of AES to ChaCha20 increases along with the file size. The larger the file size, the larger the time difference. These findings match with our previous research which state that ChaCha20-Poly1305 is faster than AES-GCM. Comparing to Cryptomator, our app encrypts files more efficiently.

Table 4 shows the test result of the asymmetric key algorithms, ECDH and RSA. Due to the benefit of ECC, ECDH can attain a comparable security level to RSA with a much shorter key length. We can see that ECDH&ChaCha20-Poly1305 has a much shorter encryption time than RSA&AES-GCM, with 434.3, 212.18, 47.18 and 22.01 times faster for files of 102.4KB, 1MB, 10MB and 100MB. Therefore, we can conclude that ECDH combination performs generally better than RSA combination in the time performance. To consider ECDH itself and RSA itself, we need to eliminate the effects contributed by ChaCha20-Poly1305 and AES-GCM. From Table 3, we can find that ChaCha20-Poly1305 performs better than AES-GCM in an increasing trend. However, ECDH combination performs better than RSA combination in a decreasing trend. From the comparison of their ratios, we can say that though ECDH performs better than RSA in the time performance, the performance is getting worse as the file size increases.

Table 5 shows the test result of ChaCha20-Poly1305 without PBKDF2. We do this extra test to explain why the symmetric key algorithm ChaCha20-Poly1305 with PBKDF2 has a longer encryption time than the asymmetric key algorithm ECDH with ChaCha20-Poly1305, which is abnormal. We find that the time spent on PBKDF2 is much longer than the time spent on the symmetric key algorithm itself, which contributes to this abnormal phenomenon. On the other hand, the time needed for EC keys generation is much shorter, which makes the ECDH encryption time short.

5.4 Limitation

There is one limitation to the test. RSA itself can be used as an encryption algorithm, which means theoretically AES-GCM can be omitted. However, RSA can only encrypt small-sized files. Therefore, we add AES-GCM to do the encryption.

6. Future Directions

There can be more cloud storages for the users to select such as Microsoft OneDrive, Egnyte, Box and so on. Also, the main cloud and the backup clouds can be decided by the users, rather than setting Google Drive as the main cloud and Dropbox as the backup cloud by default.

7. Conclusion

We have built a third-party app for cloud files encryption and sharing authentication. Cloud files are end-to-end encrypted and shared files are under the protection of device authentication. We did research on different encryption algorithms and pick the best one with the top performance in security, speed, and maturity. The encryption algorithms we chose make our app to stand out from other cloud storage encryption apps.

8. References

[1] F. TRUTA, “Google accidentally sent users' private videos to strangers in stunning 'Takeout' mix-up,” *HOTforSecurity*, 04-Feb-2020. [Online]. Available: https://hotforsecurity.bitdefender.com/blog/google-accidentally-sent-some-peoples-private-videos-to-strangers-in-massive-takeout-mix-up-22226.html. [Accessed: 10-Mar-2021].

[2] C. J. McKinney, “Personal data breaches by the Home Office soar to over 4,000 last year,” *Free Movement*, 28-Sep-2020. [Online]. Available: https://www.freemovement.org.uk/personal-data-breaches-by-the-home-office-soar-to-over-4000-last-year/. [Accessed: 10-Mar-2021].

[3] Crytomator, “Security Architecture¶,” *Security Architecture - Cryptomator 1.5.0 documentation*. [Online]. Available: https://docs.cryptomator.org/en/latest/security/architecture/. [Accessed: 10-Mar-2021].

[4] S. Gueron, A. Langley and Y. Lindell, "AES-GCM-SIV: Specification and Analysis", International Association for Cryptologic Research ePrint Archive, 2017. [Online]. Available: https://eprint.iacr.org/2017/168.pdf. [Accessed 10-March-2021].

[5] KDDI Research, Inc, " Security Analysis of ChaCha20-Poly1305 AEAD," Cryptography Research and Evaluation Committees. (CRYPTREC), Japan, 2601, 2016, pp. 2-32. Accessed on Dec. 2, 2020. [Online]. Available: https://www.cryptrec.go.jp/exreport/cryptrec-ex-2601-2016.pdf. [Accessed 10-March-2021].

[6] D. Blazhevski, A. Bozhinovski, B. Stojchevska and V. Pachovski, "MODES OF OPERATION OF THE AES ALGORITHM", in *The 10th Conference for Informatics and Information Technology*, 2013. [Online]. Available: http://ciit.finki.ukim.mk/data/papers/10CiiT/10CiiT-46.pdf. [Accessed 10-March-2021].

[7] H. Lipmaa, P. Rogaway and D. Wagner, *Comments to NIST concerning AES Modes of Operations:CTR-Mode Encryption*. National Institute of Standards and Technologies, 2000. [Online]. Available:

https://www.researchgate.net/publication/2817314_Comments_to_NIST_concerning_AES-modes_of_operations_CTR-mode_encryption

[8] D. Mahto and D. K. Yadav, “RSA and ECC: A Comparative Analysis ,” *International Journal of Applied Engineering Research*, vol. 12, no. 19, pp. 9053–9061, Jan. 2017.

[9] D. Mahto, D.A. Khan, D.K. Yadav, "Security Analysis of Elliptic Curve Cryptography and RSA," in *2016 World Congress on Engineering: Vol I, June 29, 2016, London, U.K.* [Online]. Available: http://www.iaeng.org/publication/WCE2016/WCE2016_pp419-422.pdf. [Accessed 10-March-2021].

[10] M. Gobi, R. Sridevi, and R. Rahini priyadharshini, “A Comparative Study on the Performance and the Security of RSA and ECC Algorithm,” *International journal of advanced network and application* , pp. 168–171, Mar. 2015.

[11] J. Cui, Y. Qi, B. Hong and Q. Chen, "Research on Cloud Computing Data Security Based on ECDH and ECC", *International Journal of Simulation: Systems, Science & Technology*, vol. 17, no. 35, pp. 201-207, 2013. Available: https://ijssst.info/Vol-17/No-35/paper20.pdf. [Accessed 10-March-2021].

[12] A. Langley, “Protecting data for the long term with forward secrecy,” *Google Online Security Blog*, 22-Nov-2011. [Online]. Available: https://security.googleblog.com/2011/11/protecting-data-for-long-term-with.html. [Accessed 10-March-2021].

[13] Datanyze, “Dropbox Market Share and Competitor Report: Compare to Dropbox, Google Drive, Microsoft OneDrive,” *Datanyze*. [Online]. Available: https://www.datanyze.com/market-share/file-sharing--198/dropbox-market-share. [Accessed: 09-April-2021].

[14] E. Ravenscraft, “Cloud Storage Showdown: Dropbox vs. Google Drive,” *zapier*, 19-Dec-2019. [Online]. Available: https://zapier.com/blog/dropbox-vs-google-drive/#Security. [Accessed: 09-April-2021]

[15] A. C. Pincher, “Google Drive vs Dropbox: Which cloud storage is better?,” *JotForm*, 08-Mar-2021. [Online]. Available: https://www.jotform.com/blog/dropbox-vs-google-drive/. [Accessed: 09-April-2021]

APPENDIX

Table 6. Time performance on encryption for ChaCha20-Poly1305 and AES (details)

Algorithm / file size	102.4KB (ms)	1MB (ms)	10MB (ms)	100MB (ms)
ChaCha20-Poly1305	1st: 2 2nd: 0.96 3rd: 0.86 AVG= 1.27	1st: 6.63 2nd: 5.10 3rd: 5.82 AVG=5.85	1st: 47.26 2nd: 49.70 3rd: 45.70 AVG=47.56	1st: 438.70 2nd: 421.10 3rd: 393.80 AVG=417.87
ChaCha20-Poly1305 + PBKDF2	1st: 28 2nd: 27 3rd: 28 AVG = 27.67	1st: 44 2nd: 38 3rd: 42 AVG = 41.33	1st: 108 2nd: 99 3rd: 101 AVG = 102.67	1st: 652 2nd: 643 3rd: 637 AVG = 644
AES GCM + PBKDF2	1st: 41 2nd: 33 3rd: 36 AVG = 36.67	1st: 159 2nd: 132 3rd: 131 AVG = 140.67	1st: 1106 2nd: 1117 3rd: 1049 AVG = 1090.67	1st: 10324 2nd: 10764 3rd: 10747 AVG = 10611.67

Table 7. Time performance on encryption for ECC ECDH and RSA OAEP (details)

Algorithm / file size	102.4KB (ms)	1MB (ms)	10MB (ms)	100MB (ms)
ECC ECDH ChaCha20-Poly1305 256 bits	1st: 4 2nd: 5 3rd: 2 AVG = 3.67	1st: 9 2nd: 11 3rd: 8 AVG = 9.33	1st: 59 2nd: 62 3rd: 70 AVG = 63.67	1st: 544 2nd: 545 3rd: 561 AVG = 550
RSA OAEP AES-256 GCM 3072 bits	1st: 1558 2nd: 1509 3rd: 1715 AVG = 1594	1st: 1849 2nd: 1895 3rd: 2195 AVG = 1979.67	1st: 3353 2nd: 2763 3rd: 2895 AVG = 3003.67	1st: 12120 2nd: 10819 3rd: 13385 AVG = 12108

FINAL CONSOLIDATED LOGBOOK

NAME: WONG SHING

SID: 1155109027

Week 1:

Planning:

Start to work on the ECC encryption.

Done:

Have found some related libraries of ECC ECDH encryption.

Week 2:

Planning:

To implement the auto writing and reading of SALT using one single file. The encrypted file will have its first 16 bytes as the SALT.

Done:

We have successfully implemented the operations with SALT in one single file. During encryption, the SALT is put into the first 16 bytes of the encrypted file. During decryption, its SALT is read automatically for decryption. No separated file is generated.

Week 3:

Planning:

We plan to finish the proposal. This is because we can set timelines for ourselvels and clarify our directions.

Done:

We have finsihed the proposal. We have break the works we need to do into pieces so that we have better arrange our time to implement the ECDH, user interface and evaluation. Also, we are now clear about what we are going to do and how to do them.

Week 4:

Planning:

I plan to implement ECC ECDH encryption method.

Done:

We can now generate the key pairs used in ECC ECDH encryption, the private key and public key. They will be used later for file encryption with ECC ECDH.

Week 5:

Planning:

We plan to improve the UI design.

Done:

We have changed the password input from plaintext to ‘*’. This implementation can better protect the others from stealing the password.

Week 6:

Planning:

Prepare for evaluation. Implementing AES that will be used to compare with ChaCha20-Poly1305.

Done:

Implementation of AES are similar to ChaCha20-Poly1305. Some codes in the ChaCha20-Poly1305 implementation can be reused. We have imported the suitable libraries.

Week 7:

Planning:

Finish the whole implementation of ECC.

Done:

We have finished the key pairs generation. We have successfully encrypted the encrypted file from ChaCha20-Poly1305. We can exchange the ECDH public key. We can decrypt the files with both ChaCha20-Poly1305 and ECDH key pairs. To conclude, we have implemented the whole process of ECC.

Week 8:

Planning:

We plan to finish the function of file selection from Google Drive.

Done:

We have successfully selected files from the Google Drive. We can get the filename and fileID. After selection, the files is downloaded to the phone, stored in the local file system.

Week 9:

Planning:

We plan to implement the file picker for Android local file system.

Done:

We have implemented the Android local file system search. User can select any files from the local file system. The app then will automatically pick the selected and do encrypt and upload.

Week 10:

Planning:

We plan to implement the UI of ECC, such as file selection, encryption etc..

Done:

We have added a page for ECC file selection, corresponding encrypted files are shown, and clients can select the file to select. Also, the client can select the user's public key.

Week 11:

Planning:

We plan to create a function to help the clients to share their public key.

Done:

We choose to use QR code as the method for the client to share their public key. We have done analysis on the advantages and disadvantages of using QR code.

Week 12:

Planning:

We plan to further implement the QR code functions for the client to get the public key of the receiver though QR code.

Done:

We have implemented a QR code Scan function and a QR code from gallery function for the clients to add the receiver’s public key. Our code can read the QR code and generate a receiver public key file in a specified folder. Later, the client can choose the receiver public key via an array list shown to them.

Week 13:

Planning:

So far, we have roughly finished all the implementations. Now, we plan to start to work on the final report.

Done:

We have finished the abstract, introduction, methodology and results. Now, we are working on the discussion part to describe out test results and explain the details. After that, we will be working on the future work and conclusion part.

Week 14:

Planning:

It is nearly the end of the semester and we are ready to publish our app. We will do a final check on the codes to make it clean and efficient. After that, we would like to make it publishable.

Done:

We have made all the variables names semantically readable and all codes clean and efficient. The codes are in production mode. We have generated APKs of our app for others to test on it. APK form is for sharing the app.