Login or registration

When you register or sign up for PriSecFileStorage service, it generally uses public key authentication that was described in the MFA folder's FIDO document by using pure randomness in generating the private and public key for the digital signature key pair. This way, server can't leak your password/passphrase. All the data that was stored on the server was all considered as publicly disclosable data. The FIDO document describes the server only uses your public key to authenticate users.

If you lose your private key that's located in either "Application_Data\Server_Directory_Data\Storage_ID\rootSK.txt" or "Application_Data\Other_Directory_Data\Storage_ID\SK.txt", you can't login to your storage anymore.

Symmetric encryption generation, usage and storage

PriSecFileStorage uses total endpoint encryption with pure randomness. Symmetric encryption keys are generated by relying on pure and secure randomness mechanism. These keys are generally stored only in your own device once they are generated.

Unlike normal privacy or confidential file storage, PriSecFileStorage does not involve any user secret such as password or passphrase on service provider side. Normal privacy or confidential file storage requires user to submit their password or passphrase so that they can perform encryption and decryption as they see fit. This was not considered as privacy or confidential respecting at all because you temporarily give them your password or passphrase.

Due to the fact that it uses total endpoint encryption or in simpler words perform encryption and decryption on your device such as laptop/desktop or mobile phone, any symmetric encryption keys that are generated and stored on your device will be responsible for encrypting or decrypting the file. If you happen to lose them, there's nothing I can do.

This is the reason why you have to do backup to those symmetric encryption keys. They are generally located in either "Application_Data/Server_Directory_Data/Storage_ID/Encrypted_Files/Random_File_ID/Keys" or "Application_Data/Other_Directory_Data/Storage_ID/Encrypted_Files/Random_File_ID/Keys" folder.

Digital Signature

The files will be encrypted and then signed with a random generated private key. The encrypted and signed file will then be sent to server. This way it’s not possible for the attacker to change the encrypted file without any notice.

Server or I was not the party to verify the encrypted signed file. Hence, the public key was never send to server as well.

You may send to your friend or other trusted party but never the service provider.

The digital signature key pair that was used in signing the file can be found in either “Application_Data/Server_Directory_Data/Storage_ID/Encrypted_Files/Random_File_ID/ED25519SK” or “Application_Data/Server_Directory_Data/Storage_ID/Encrypted_Files/Random_File_ID/ED25519PK” or “Application_Data/Other_Directory_Data/Storage_ID/Encrypted_Files/Random_File_ID/ED25519PK”.

You may notice the presence of “ED25519SK” or “ED25519PK” folder and any file that contains the word of either “SK” or “PK” is all digital signature.

ED25519 is a state of the art digital signature algorithm that was based on the famous named elliptic curve, Curve25519.

You may also heard of RSA (Asymmetric encryption/digital signature), DSA(Digital Signature Algorithm), ECDSA (Elliptic Curve Digital Signature Algorithm), ED448 (Edward Curves [Elliptic Curve] Digital Signature Algorithm) or EdDSA.

MAC (Message Authentication Code)

If you recall what I stated in stream cipher, they are too risky to use on their own. Hence, when we send encrypted message that has been encrypted through stream cipher, we usually append or stick MAC in front or behind of the encrypted message.

Simply put, the idea of MAC was to prevent the stream cipher encrypted message from being able to alter.

Modern industry MAC algorithm consists of HMAC-SHA1, HMAC-SHA256, HMAC-SHA512, Poly1305, GMAC.

In the software, you will generally see “Poly1305” or “GMAC”. Poly1305 was used in conjunction with XSalsa20 and XChaCha20 whereas GMAC was commonly seen in AES-GCM.

Symmetric Encryption Algorithm

In most cases, you may have heard of AES (Block Cipher), ChaCha20 and Salsa20 – Stream Cipher.

Modern industry standard typically use AES in stream cipher format or officially known as AES-GCM as it’s much faster compare to using AES in block cipher format. However, the AES-GCM you will be seeing in this software is hardware accelerated version. It generally requires the CPU chip with the minimum version of Intel SSSE3 chips or similar other brand chips. If you happen to use it on your device, your device may not support hardware accelerated version of AES-GCM.

If your CPU chip does not support running hardware accelerated version of AES-GCM, you will need to consider whether you want old industry standard stream cipher or new industry standard stream cipher.

Older industry standard stream cipher refers to a stream cipher that has goes through a lot of testing and rightful alteration. It’s more secure and enduring in this way. If you prefer older industry standard stream cipher, then your choice would probably be “XSalsa20Poly1305>XChaCha20Poly1305>AES256GCM”.

Newer industry standard stream cipher refers to a stream cipher that goes through slightly lesser testing and rightful alteration. It’s less secure and enduring in this way. If you prefer newer industry standard stream cipher, then your choice would probably be “XChaCha20Poly1305>XSalsa20Poly1305>AES256GCM”.

When you look back the history of Salsa20 or ChaCha20, you will realized that ChaCha20 was published 3 years later (2008) than Salsa20 (2005). ChaCha20 was designed to be more secure and faster compare to its ancestor – Salsa20. Just like wine, the older the better. However, at least 10 years have passed, one may argue that both of them are safe to use now.

In recent video made by Computerphile, Dr Mike Pound stated that ChaCha20 was chosen to be an alternative to AES. Not to mention that Google the big tech company also pushes to encourage people to use ChaCha20. Open source operating system like Linux also uses ChaCha20 in a way to produce nearly perfect random number.

The choice is really up to you the user, if your device can't support hardware accelerated version of AES-GCM, you may want to choose an alternative to AES which is a new rising symmetric encryption algorithm standard – ChaCha20 or you can just stick to Salsa20, the old and more tested version of symmetric encryption algorithm – Salsa20.

Each of them was considered safe and secure to use. It's just depending on you the user whether you want to choose an older wine or a newer wine but faster.

How was file encrypted?

For a minute we will use digital signature algorithm – ED25519 and stream cipher with MAC – XSalsa20Poly1305.

Every time you select a file, its content by default will be separated into parts. For example, the file size was 15 MB. The file will be separated into 3 parts with each 5 MB in size.

The system will randomly generate ED25519 public and private key and symmetric encryption key.

Each part of the file will be first encrypted with the XSalsa20Poly1305 then signed with ED25519 private key. The whole signed encrypted file was then sent to the server. All symmetric encryption key, the ED25519 private and public key was never sent to the server.

Encrypted_File = ED25519(Encrypted_Text||MAC)

The "||" symbol refers to append MAC to the encrypted text.