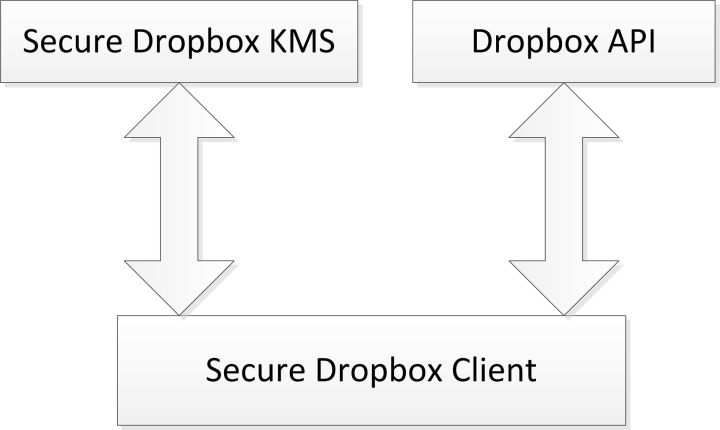
# Chapter 3. Design

Secure Dropbox is designed as a client end encryption tool over Dropbox service. The main idea is providing a file encryption service with symmetric cryptography that the key is not known to Dropbox and in the meantime the file encryption key is well protected with asymmetric cryptography so that it is secure to be shared. It is proposed to against the current way of how Dropbox handle users’ file content security and potential information security risks due to this mechanism especially from internal Dropbox as they could easily get across to these file content. Also the security against external attack to Dropbox has been fortified since files are encrypted twice from Dropbox and Secure Dropbox respectively. In terms of design, Secure Dropbox would be a C/S architecture that the server end would work like a key management system (KMS) which mainly processes key management related requests and storage request while the client end running on users host computer would perform the most resource consuming cryptography related computations. Secure Dropbox would perform file operation like uploading, downloading and sharing via Dropbox API or the Dropbox official client which is built with Dropbox API as well. User interface of Secure Dropbox will be designed as a file system interface. All file operation is no more than CRUD which indicates that although cryptography procedure is literally everywhere, users would not recognize them as these procedure is automatically played accompany with CRUD operations. The very security of Secure Dropbox service would be based on a proper usage of Secure Dropbox account information by users. Two application modes would be provided: one as regular mode when there is Internet access and another one as local mode when there is no Internet access. Different operation permission schema is granted to users.



This chapter will discuss about some major design decisions and implementation challenges and more potential features of Secure Dropbox project. Firstly, the overall design Secure Dropbox will be proposed. Secondly, the cryptography application mechanism to guarantee the security of Secure Dropbox will be expressed and justified. What is more, the reason of making choices about ways of performing file system related operation, integration with Dropbox and platform on which to construct the application will be discussed with regard to building a platform independent application. Additionally, the decision made in terms of building a C/S architecture application would be explained. Some other design features are more intuitively understandable by explaining from implementation aspect so more detail of those part would be proposed in next chapter or with just brief mentioned in this chapter. Last but not least, the design decisions made to Secure Dropbox or other principles are not only Dropbox oriented specifically but could be adopted when designing any application that protect users file in the cloud and provide a reliable file in cloud sharing system.

## 3.1 Application of Cryptography

Although Dropbox claims that they are using modern encryption methods on data transmission with Secure Socket Layer (SSL) and performing AES 256bit encryption to store users’ data, which seems perfect when defending against external attack but only on condition that the hacking does not come to key management service (KMS), these mechanism make no sense of protecting users’ data from internal attack, which is usually performed by employees of Dropbox those has access permission to KMS. The behavior, or say the ability, of accessing to users’ data has been admitted in their terms of service [1] that Dropbox will remove the encryption from file data and deliver them to law enforcement on government’s requirement. Because of internal attacker’s proficient understanding of Dropbox system or some rampant, improper access permission configuration, some internal attack could even be done unconsciously by users. These security concerns have been certified that several internal attack instances are reported. Client end encryption is one of those alternatives recommended by Dropbox if users are more willing to conceal their data and care less about losing some Dropbox features like version control or data recovering [1]. More importantly, while this potential way protect user’s own data from internal Dropbox attack to some extent, it actually disable the file sharing infrastructure provided by Dropbox totally.

Anyway, cryptography is still going to be the very insurance to the security of Secure Dropbox. Generally, Secure Dropbox is designed as a client end encryption tool which makes Dropbox internal attack impossible and also provide cipher file sharing mechanism. To achieve this goal, Secure Dropbox would apply a combination of symmetric cryptography algorithm and asymmetric cryptography algorithm to secure the file storage. The symmetric ones like AES or DES, would be used on file content encryption given the fact that they are much faster than asymmetric cryptology and enough security offered as another layer upon the Dropbox AES 256bit Encryption. The asymmetric cryptology is used for protecting the symmetric encryption key and providing a secure key transmission procedure. In the following example, the whole procedure of secure storage would be illustrated with AES and RSA as instance of symmetric and asymmetric cryptography:



Figure.1 Encrypted File Storage / Reading

In file encrypted storage procedure, plaintext is ciphered in AES with a random generated AES file key. The doc key is later ciphered as well in RSA with file owner’s RSA public key.

In encrypted file read procedure, firstly the ciphered doc key Ksecure should be deciphered in RSA with file owner’s RSA private key. Having retrieving the raw doc key K, the cipher could be encrypted in AES with K.

The procedures of sharing from Secure Dropbox users is mainly about ciphering the doc key K and transferring it to the sharing recipient. K should be ciphered by file owner but could only be deciphered by sharing recipient but no one else. This is a typical scenario for asymmetric cryptography application. In the following example, Alice wants to share a file with Bob:

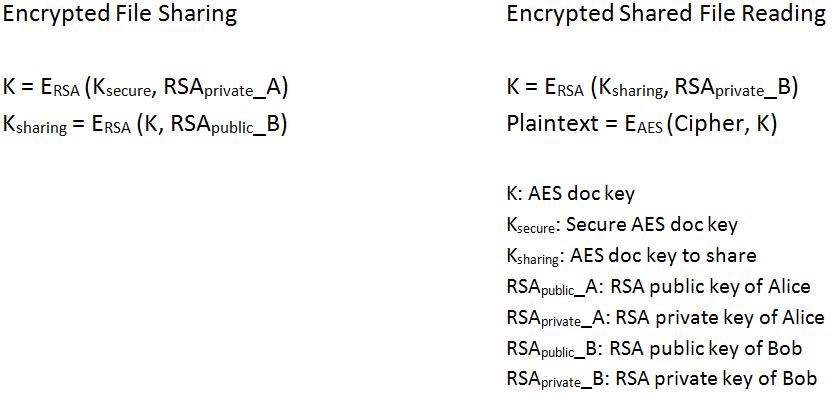


Figure 3.2 Encrypted File Sharing

Sharing procedure as shown above actually does not perform an operation on encrypted files but only ciphering the key for sharing purpose. The doc key is stored in secure way at the very beginning. To get the raw doc key, Alice has to decipher Ksecure in RSA with her own RSA private key. In order to enable Bob deciphering the key ciphered by Alice, She should encrypt the raw doc key K in RSA with Bob’s RSA public key, which is open to all the user in Secure Dropbox system. Ksharing is generated in this way and then transmitted to Bob via secure methods. After Bob receiving the Ksharing, he deciphers it in RSA with his own RSA private key, gets the raw doc key and the plaintext of shared encrypted file could be retrieved by decrypting in AES with the raw doc key.

With regard to how to manage the RSA key pair of Secure Dropbox user and doc key of each file, a detailed design description would be given in the coming section.

## 3.2 KMS

In Secure Dropbox application scenario, it is in essence that there is an infrastructure posting all Secure Dropbox users’ RSA public key for sharing sponsor to obtain during the encrypted file sharing procedure. In addition, given the essence of Secure Dropbox (a client end Dropbox encryption tool) and Dropbox’s cloud storage nature, the requirement of everywhere use should be meet. It means as long as users could get access to Dropbox, they are allowed to use Secure Dropbox service like download user’s own doc key chain and RSA key pairs. To achieve the goal, besides a local copy of essential data like Secure Dropbox users’ authentication information, RSA key pairs, doc keys and file sharing information, a permanent copy should be stored on server which is able to be accessed anytime anywhere. Such a key management service should be built in Secure Dropbox system.

However, in order to avoid same mistakes that made by Dropbox in terms of encryption mechanism, some improvements should be made. Dropbox stores file encryption key in its own database makes it not an accredited storage service to users. To make Secure Dropbox reliable, the authority of decrypting file should be only granted to users but no one else even the KMS administrator. A potential alternative is storing all sensitive data (e.g. RSA private key and doc key) on the server confidentially so that the everywhere usage could be realized safely. Therefore, the only entry (assume it could be something like a user token or password) of decrypting the all these sensitive should never be stored on server but only held by users themselves.

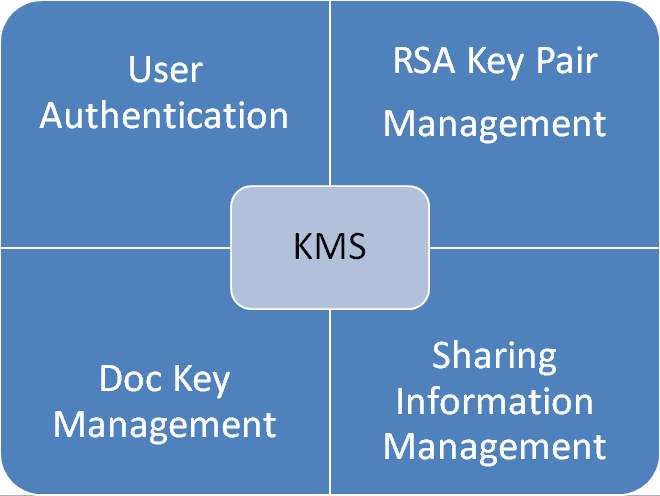


Figure 3.3 KMS Architecture

### 3.2.1 KMS User Authentication

User authentication information is basic component in any application instance as long as access control is required. The classic way of how web application saves user’s authentication could be adopted: for the minimum usage, a plaintext username and a hashed password should be stored. The authentication procedure is actually matching the username and password hash value. The hashed password is generated in the client and only the hash value with algorithm indicator prefix will be uploaded to KMS and stored. To avoid performing any suspicious behavior, plaintext password should be hashed in client once it is input and uploaded for authentication where plaintext password never involved. The matching hash value could be performed at client as well but it requires KMS interface for user to fetch hash algorithm’s salt, iteration times and hash value.

### 3.2.2 KMS RSA Key Pair Management

RSA key pair could be generated either on KMS end or client end although it would potentially bring about better user confidence when everything is generated in client end and uploaded only after getting encrypted. In KMS, RSA public key is stored in plaintext so that it could be reached by users who want to process the doc key before sharing the file. RSA private key should be stored in cipher which is encrypted in symmetric cryptography with user’s own password or token as encryption key directly after being generated in client end. RSA public key is open for file sharing’s convenience. Any request about fetching Secure Dropbox users’ RSA public key is allowed.

### 3.2.3 KMS Doc Key Management

For each file encrypted by Secure Dropbox, a unique document key should be generated. Before getting uploaded to KMS, it is a must to encrypt the document key in RSA with file owner’s RSA public key. Document’s name could be stored in plaintext for easy indexing. In this way, any operation on this encrypted file, no matter reading or sharing, could only be initiated by file owner since in both procedures it is necessary to get the doc key which is only accessible by people who has the RSA private key to perform the decryption in corresponding to the encryption with RSA private key from same key pair.

### 3.2.4 KMS Sharing Information Management

The key component of sharing data is still the processed doc key. The processing procedure should be performed on client just as any other sensitive data. The sharing recipient should be able to get the processed key and decrypt it with own RSA private key. Besides it should include sharing metadata generated by Dropbox sharing API: a URL indicate the entry of get the raw file content and an sharing expiration timestamp indicate when would the URL would be closed for access.

### 3.2.5 Local KMS

There are two modes for Secure Dropbox: regular modes when Internet access is available while a local mode when Internet access is not available. A local copy of KMS information related to Secure Dropbox user would be generated for local usage like user authentication information, RSA key pair information and doc keychain. Since there is no Internet access, User can neither share a file with other Secure Dropbox users nor read shared file from other Secure Dropbox user. But user’s own file are still accessible as they are stored in the local file system as well given the decision about using Dropbox official client as a file container. A local RSA key pair and doc keychain makes it possible to decrypt any encrypted files. Apparently this local copy should be protected and encrypted as well.

## 3.3 Right Cryptography Algorithm

### 3.3.1 Symmetric Cryptography

In the symmetric (private-key) cryptography, encryption and decryption are performed with same key [2]. Accordingly, encryption key sharing becomes the precondition of the sharing symmetrically encrypted information. Theoretically, as it is significantly faster in comparison with asymmetric algorithm with same security level given their different nature, symmetric encryption algorithm is usually used for relatively big data protection.

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [3]. NIST claims that AES with 128-bit keys provides adequate protection for classified information up to the SECRET level definition. Increasing AES applications have changed to the AES-256 in which more rounds hash and longer key length are applied and fulfill a TOP SECRET level by definition (Both definition of SECRET and TOP SECRET are advised by CNSSP-15 [4]). Theoretically there are 3.4 X 1038 combinations to guess for AES-128 while this number is squared and comes to an incredible 1.1 X 1077 for AES-256.

As Dropbox is using AES-256 for Encryption in transit and Encryption in rest, apparently it would be ideal to deploy an AES-256 encryption in Secure Dropbox for file encryption as well. As claimed by NIST [6], the encryption strength of AES-256, which has an equivalent security level as RSA, would be sufficient until 2030. In Secure Dropbox, AES-256 would be the most direct protection on all data include encrypted file storage, encrypted local storage of user’s document key chains and other confidential data in server end as long as content conceal is required.

However, since the guaranteed security totally depends on a proper key usage and storage, it is highly risky to transmit the unprotected file encryption key via security unreliable media like Internet. To deal with this problem, despite of those application layer protocols for secure data transmission like SSL, hybrid encryption which combines the symmetric encryption and asymmetric encryption would be better way to directly protect the symmetric encryption key against risks during key exchange.

### 3.3.2 Asymmetric Cryptography

Practically, the security of cryptography nowadays is no longer guaranteed by concealing the cryptography algorithm itself but lies on the encryption key strength mechanism of key protection. While sharing the symmetrically encrypted data inevitably involves key exchange on unreliable public tunnels where interception and distortion are happening, the asymmetric cryptography turns out to be a better alternative.

In asymmetric cryptography, the encryption and decryption are performed with mathematically linked public key and private key respectively. Generally speaking, public key is for encrypting the plaintext while private key is used for decryption purpose. The key pair is generated by a trusted PKI and the key pair, especially the private key which is the only entry of decryption, is distributed far less often than symmetric encryption application scenario and which essentially reduces the security threats brought by frequent key exchange. As prerequisite of encrypted information sharing, the public encryption key is widely distributed or accessible by anyone who wants to activate information sharing, while the private decryption key is known only to the encrypted information recipient. As a result, there is only transmission of encrypted data which could be considered as secure rely on the strength of asymmetric cryptography algorithm or the key length but without any key exchange. Nevertheless, due to the different mathematical nature with symmetric cryptography, asymmetric algorithm is remarkably less efficient there by its most application is often limited to small data encryption or as a component in hybrid encryption. For example, to share a large encrypted data, the data content is encrypted in fast symmetric cryptography such as AES while the AES key is protected with asymmetric cryptography before transmission.

RSA is an [algorithm](http://en.wikipedia.org/wiki/Algorithm) for [asymmetric cryptography](http://en.wikipedia.org/wiki/Public-key_cryptography)  based on the presumed difficulty of [factoring](http://en.wikipedia.org/wiki/Factorization) [large integers](http://en.wikipedia.org/wiki/Integer). As claimed by RSA Security, the strength of RSA with certain length is equivalent or comparable to symmetric cryptography with different length. The correspondence is listed as follows:

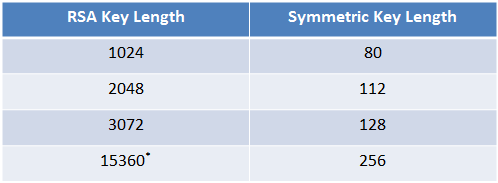


Figure 3.4 Strength Equivalence [5] (\*Advised by NIST [6])

RSA Security also claimed that 1024-bit keys are likely to become unsafe sometime between 2006 and 2010 while the 2048-bit keys are sufficient until 2030. Thus the RSA key length of 3072 bits is required since 2030 [5]. RSA with 2048bit or longer key length should meet the requirement of file encryption key security of Secure Dropbox.

### 3.3.3 Cryptographic Hash Function

Cryptographic hash functions basically perform irreversible encryption on an arbitrary length plaintext and returns a fixed length cipher. The hash digest is fixed for same input but tremendously changed as long as there is changes happen to content, no matter how trivial it is [7]. These one-way hash functions provide a rapid method for detecting any changes to the content and are also used for generating identical digest of content [7]. In web application context, user’s confidential information is usually stored in the form of hash digest but not plaintext in case of being exposed when server hacking happens.

Since hash function by definition has same output with same input, hash collision attack is easy to perform. For example: Alice set her password as 123456 while Bob coincidentally set his password as 123456 as well. According to the nature of hash function, a same password hash value would be generated and stored. If Alice happens to get the password hash table that stored in server, she might recognize that Bob’s password hash value is exactly the same with hers and further recognize that Bob could have the same password as Alice does:

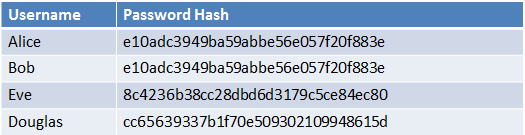


Figure 3.5 Hash Table

What is more, if Alice has something like ***whole collection of most frequently used password*** and the hash algorithm is open, she could try to hash each password in the collection and match to see if same hash value exists. Accidentally she might find that the password hash value of Eve could be achieved by hashing 654321 with the same hash function that the server is using which means Eve’s plaintext password is known to Alice as well.

To protect password hash value against hash collision attack, a random generated salt and configurable hash function iteration time could be applied when performing the content hashing. Randomly generated salt acts as a part of content to make same content identical. The stored hash value is the result of hashing the primary content and salt. Also the adoption of different iteration with different round times leads to a more sophisticated mapping relationship between plaintext and hash value. The following example illustrates how a hash table with salt and iteration information stored in server:

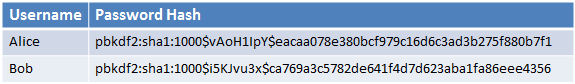


Figure 3.5 Hash Table with Salt, Iteration and Algorithm Indicated

As shown above, there is additional information attached to the hash value. Take first record for example:

* pdkdf2 indicates the where the hash algorithm is implemented and used from
* sha1 indicates the algorithm of how is the following hash value generated
* 1000 indicates there are 1000 iterations when performing the hashing
* vAoH1lpY is the salt added to the plaintext password before it is hashed.

The hash value to be matched should be generated with above parameters. In this example, the SHA1 which generates a 160bit long digest of content could be considered as a sufficient secure hash function since it is still widely used in main stream security protocols like TLS and SSL.

### 3.5 Integration with Dropbox

Since Secure Dropbox by nature is a Dropbox user oriented client end file encryption tool, the usage of infrastructure provided by Dropbox should be still remained and available to Secure Dropbox users. Dropbox provides kinds of APIs for Dropbox developers. For example, The Dropbox Core API, which is advised by Dropbox is ideal for server-based apps, provides programming interface to read and write to Dropbox and actually the most direct way to access Dropbox. Also it includes some advanced functionality interfaces like content search, version controls, and restoring files service for developers to performing low-level controls. Additionally, the Dropbox Sync API, which provides file system-like programming interface, encapsulates functionality implementations like syncing and notification of remote changes and orients to mobile platform usage. Since in Secure Dropbox, not only file synchronization operation like uploading and downloading, but also the some low-level operation like file sharing, sharing url generation and file metadata check are required, the Dropbox Core API is taken into consideration. To use Dropbox Core APIs, an access token obtained during Dropbox OAuthentication is essential. Basically it allows Secure Dropbox, a third-part application, able to use Dropbox API with granted permission but never get disclosed with any Dropbox user account information. The implementation details about how to perform the indirect authentication will be proposed in the implementation chapter. Some key Dropbox Core APIs are listed as follows:

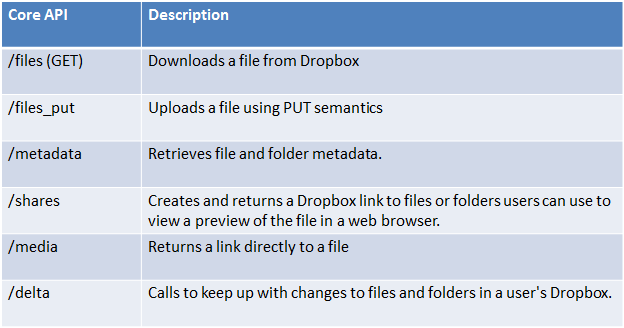


Figure 3.6 Key Dropbox Core APIs Functionality Description

Except exploiting these Dropbox APIs as advised, there is a more tricky way to use Dropbox service – get everything synchronized through the Dropbox official application. Using Dropbox on host computer is just like using any other folder in the file system, except the files you [drag](https://www.dropbox.com/help/90) or copy into it will be automatically synchronized online and to any other computers or mobile devices [linked to your account](https://www.dropbox.com/help/4) where there is Dropbox application installed as well. It is essentially a way of using the Core API indirectly because it is exactly the underlying interface for all of their official applications and [SDKs](https://www.dropbox.com/developers/core/sdk), just as claimed by Dropbox. The story with regard to file operation could be tremendously simplified if Secure Dropbox use Dropbox client application as target folder and perform all file operations on it. Correspondingly, every encrypted file write into Dropbox client folder would be uploaded automatically but without any Dropbox API invoking. A commercial software product with same design and implementation in terms of the hybrid usage of Dropbox APIs is Boxcryptor [8]. Boxcryptor has a driver level encryption which is more efficient that user mode encryption. However, the nature of client end encryption disables the sharing service infrastructures provided by Dropbox since file shared that way would be nonsense to human reader since data are encrypted. Boxcryptor made their own sharing service with partially depends on Dropbox API service like getting raw content of encrypted file via /media interface and sharing after decryption.

Although Dropbox client folder eases the complexity when design the file operation module of Secure Dropbox, the file sharing feature of Dropbox client has been disabled. To analyze from API invoking’s perspective, /share function returns the url refers to file or folder entity encapsulated with html information so there could be a rendered display of certain entity or, with /media function it returns the url refers to a raw file content text. Apparently, if a file has been encrypted by Secure Dropbox and uploaded to Dropbox, it would be of course accessible by authenticated user but not meaningful until these encrypted data has been through the decryption service provided by Secure Dropbox who knows the scheme of encryption algorithm and where to get the file encryption key from. A potential solution to such a problem is a combination usage of Dropbox client and Dropbox API. For instance, the shared information could be accessed by firstly get the encrypted raw data via /media interface and get these data decrypted in Secure Dropbox while other synchronization operations are still based on Dropbox client just as manipulate files on local file system. Another concern derives from the lack of access controls on Dropbox application. The permission to Dropbox application could go wrong in the following scenario: Alice logins to Secure Dropbox service with her Secure Dropbox account and performs some file operations at her laptop. Bob comes to Alice and asks for a temporary usage of her laptop. When Bob logins with his Secure Dropbox account and uses Secure Dropbox, the file operations are actually performed on Alice’s Dropbox account. Since Dropbox does not open the API to reset the Dropbox client account information, an advisory, potential and unreliable solution is manually advising Secure Dropbox user to switch user account of Dropbox client before using Secure Dropbox.

## 3.6 C/S or B/S

As so long as there is cryptography operation happens in the server end, there is inevitably encryption key involved and which also indicates that the server is allowed to do anything with the encryption key like storing or distributing it. So, to make Secure Dropbox security confident to users, KMS of Secure Dropbox is by definition not allowed to involve any cryptography procedure. Apparently a B/S architecture, where client in the browser end is designed to be as light as possible while the server is almost fully responsible for performing all the computations, is not considered as a reasonable decision. It could be the foremost reason why a C/S designed architecture is more appropriate. In C/S architecture, to reduce the workload of single server, client is usually assigned with more tasks and only the processed data will be submitted to server while in B/S the browser works as the client but usually designed to be an IO interface. Besides the key involvement issue, in Secure Dropbox, cryptography which is literally everywhere used cost lots of computation resources especially when data to be processing is huge. All these procedure done by client end could significantly reduce the server’s workload. Given this mechanism, KMS’s work has been simplified to only processing IO request and performing local storage procedure. Where the encryption potentially involved in Secure Dropbox is illustrated as follows:

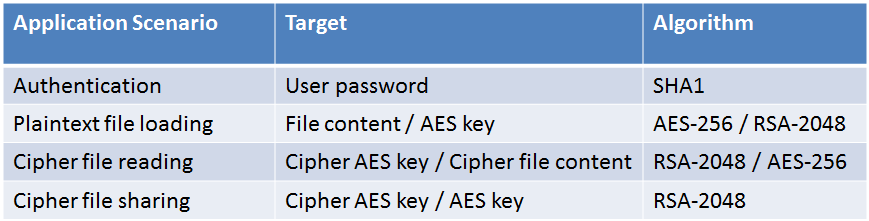


Figure 3.7 Cryptography Application Scenarios in Secure Dropbox

[1] <https://www.dropbox.com/terms/#security>

[2] Bellare, Mihir, Anand Desai, Eron Jokipii, and Phillip Rogaway. "A concrete security treatment of symmetric encryption." In Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on, pp. 394-403. IEEE, 1997.

[3]  Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001. Retrieved October 2, 2012.

[4] CNSS Policy No. 15, Fact Sheet No. 1 National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information June 2003.

[5]  [TWIRL and RSA Key Size](http://www.rsa.com/rsalabs/node.asp?id=2004)

[6] NIST Special Publication 800-57 Recommendation for Key March, 2007 Management – Part 1: General (Revised) Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid

[7] Merkle, Ralph C. "A fast software one-way hash function." *Journal of Cryptology*3, no. 1 (1990): 43-58.

[8] <https://www.boxcryptor.com/>