October 1: The Plutus Filesystem

Scribe: Tobias Boelter

October 12, 2015

Abstract

In this lecture we discuss Plutus, a cryptographic storage system that enables secure file sharing without placing much trust on the file servers.

We will see the concepts of filegroups, that enable scalable key management, discuss concepts on rights revocation and see how Merkle trees are used to efficiently update signatures.

1 Threat Model and Security Goal

The setting in Plutus is the following: There is a server where the file system is stored. Clients communicate over a network with the remote server to access files. The system offers a discretionary access control (DAC), allowing a file-owner to specify which clients can read the and which clients can additionally modify a file.

The thereat model is that the server is entirely untrusted. This can be for example because it is operated by an external untrustworthy organization or because an attacker could compromise it through software vulnerabilities. The clients are trusted with the files they have access to, but they may also behave maliciously and try to access or modify files the do not have access to. Clients can also collude with the server.

The goal of Plutus is that each client should only get access to files he is allowed to. And the server, even if he is not following the protocol, should not be able to read or modify data. The system is similar to SUNDR, but additionally provides confidentiality. To (partially) achieve this goal, Plutus combines elements from security engineering and cryptography, which makes it an interesting system to study.

When compared with SUNDR, Plutus in one regard has a weaker security guarantee, because it does not achieve fork-consistency. Generally, nothing prevents the server from forking it's internal state and running two different copies of the protocol with two different clients. In SUNDR, after a fork the server is not able to merge again in the future but in Plutus the server can actually merge changes after a fork.

Note that in the original Plutus design, DES was used as a block-cipher. In a current implementation it should be replaced with a modern block-cipher like AES, since there are known vulnerabilities in DES.

Plutus assumes the existence of an efficient key-distribution mechanism. (This assumption is at least questionable).

2 Filegroups

The file system might have already grouped files into groups through the directory hierarchy. Plutus adds another grouping, namely filegroups. A filegroup is a set of files that have the same permissions. Files in one file group do not have to be in the same folder. The filegroups are used to reduce the amount of keys that the system has to handle significantly. Plutus stores only one key per filegroup instead of one key per file.

3 Lockboxes

Plutus stores a set of keys per filegroup in a datastructure called lockbox. Each lockbox contains a bunch of DES-keys for symmetric encryption and a RSA keypair (sk, pk) for signing. The writers, i.e. clients that have write access to a file, of the file will receive the RSA keypair. The DES keys are only given to clients that are allowed to read the file (readers).

In the original Plutus design, every block is encrypted under a different key because the authors did not want to leak to many plaintext/ciphertext pairs. This is because at the time of the design, new blockciphers were frequently proposed and subsequently broken. Many of the attacks required a somewhat large number of known plaintext/ciphertext pairs so the authors wanted to mitigate this attack vector if their underlying block-cipher gets at some point broken with this sort of attacks. However, today it is assumed that a proper blockcipher must be resistant against attacks even with the availability of unlimited plaintext/ciphertext pairs and if security does not hold in this case, the blockcipher should be considered inherently unsecure. Therefore in a modern design, one would only store one key per filegroup and not one per block. Going back to the Plutus design: All the DES-keys are put into a lockbox. The lockbox is then encrypted under a lockbox key that is distributed. This allows adding later more keys to the lockbox without having to notify each client of the new key.

4 Merkle Trees

A Merkle tree is built over all the files in one filegroup. It is built over the encrypted files. Note, that if it was done over the plaintexts, this would have two weaknesses. First, it allow a dictionary attack on the blocks and second a hash function by definition does not guarantee not to leak information about the plaintext.

The Merkle root is signed with the RSA secret key.

5 The Different FS-Operations in Detail

To perform a **read**, the client fetches the encrypted blocks corresponding to a file from the server. The mechanism for how to identify the filename, which should also be encrypted, is not further specified. The client subsequently fetches the lockbox and decrypts the lockbox using it's lockbox key. It computes the hash on the encrypted file and asks the server for the merkle path to verify the integrity of the fetched file by comparing it against the signed merkle root and checking that the merkle root was correctly signed.

To **create** a new file and **give access** to it, new keys are generated, the file is encrypted under

those keys and the keys are put into the specific lockbox. If the lockbox does not exist, i.e. the corresponding filegroup does not exist, a new lockbox-key and lockbox are created and the lockbox key is distributed. It is not necessary to run a separate key-distribution mechanism for those keys because clients can simply access them with their lockbox key.

To **write** a file that already exists, the filecontent is encrypted with its existing keys and the merkle hash subtree for this file as well as the path to the merkle root are recomputed and the merkle root subsequently signed. The server should also check the access control to prevent malicious clients from modifying files. Of course, the security of the system should not rely on this check, but if the server does not perform the check, he can later be blamed for effectively deleting files as these changes would be detected by the other clients and the files subsequently rendered as invalid.

6 Security Guarantees of Plutus

Plutus achieves confidentiality, only authorized users can see the file content. Regarding integrity, freshness is not guaranteed. However it is guaranteed that a file served by Plutus was valid at some point in time. Users can not be certain that they see each other updates. Remember that SUNDR achieves fork-consistency, which is strictly stronger.

7 User Revocation

A practical system needs a mechanism to revoke users access to files. That means that a user that was given access to a filegroup in the past now is no longer given access to that filegroup. Ideally all files were encrypted with distinct keys, signed with distinct keys and all those keys were distributed. Because this is impractical, Plutus does revocation instead. That means it is fine that the user continues to see the data but will not see updates of the data. Files are only re-encrypted with new keys when updates are made to files. This can lead to a situation where many different files are encrypted with keys from different versions of the lockbox. This is where the idea of key rotation comes in. Just keep a lockbox for the latest version. If s.o. then wants to gain access to a past lockbox he uses key rotation to get the keys. However, if he wants to gain access to a future version of the lockbox, he is not able to efficiently do so.

8 Key Rotation

For the newest version b of the lockbox key For each lockbox we store another RSA keypair $sk = (N, d), pk = (N, e)$ with the property that $a^{ed} = a \mod N$. Version one of a key k_1 is just the key, generated by a key generation mechanism. The ith version is defined as k_{i-1}^d. Only the owner of the RSA secret key can compute newer versions of the key but given the public key, everybody is able to compute old versions of the key.

When the keys generated in this fashion are used as keys for the symmetric encryption scheme, this requires a stronger hardness assumption on this scheme, namely that the symmetric encryption scheme is secure under related key attacks. It's disputable if this assumption is reasonable. Standard assumptions only assume security when keys are drawn uniformly random.

9 Open Problems

It is assumed that each client can exchange keys securely with each other client and also that each client knows which client can read or write a particular file. In practice key distribution can be done by using a PKI with trusted certificate authorities (CA),

Another problem is: What if somebody loses their keys? Is the access lost forever? Secret share the key? ....

Anther problem in practice is for instance just copying a key from a laptop to a smartphone. This can for example be done through a trusted channel like a wire and a USB port. Or through encryption on an untrusted channel where the key exchange is verified through visually displaying a fingerprint of the key. This can for example be a visualization of the key or a short hash that is assumed to be unbreakable within the short time window of the key exchange process.