CoverCrypt

Full Documentation

Version 2.0

Date: 28/07/2022

Authors: Malika Izabachène (Chief Cryptographer), David Pointcheval (ENS - Cosmian Advisor), Bruno Grieder (CTO).

Table Of Contents

1 Introduction

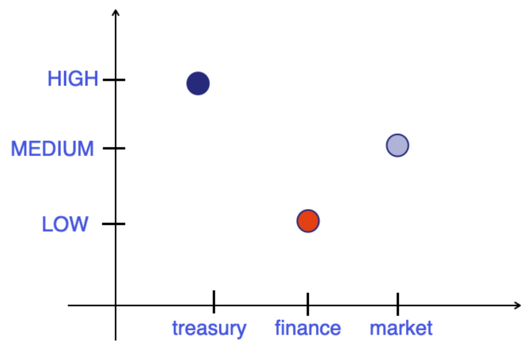
CoverCrypt is a multi-user encryption solution which provides access rights to users with respect to an access policy. It has been proposed as a more efficient alternative to Key-Policy Attribute-Based encryption sheme where the policy over attributes can be expressed as a union of users' rights. In this document and for CoverCrypt, we will use sets (a.k.a partitions) in place of attributes. We will explain the correspondance in the first section.

This documentation specifies the procedure for:

- managing users keys and access rights,
- encrypting messages with respect to target rights,
- decrypting the message when users' are granted rights for decryption.

2 Example - hierarchical axes

The figure below illustrates a hierarchical policy where domains: treasury, finance, market; and security level in increasing order: HIGH, MEDIUM and LOW are on x- and y- axes respectively.



- S_1, S_2, S_3 with $S_1 \subset S_2 \subset S_3$ are associated to finance with levels LOW, MEDIUM and HIGH respectively;
- S_4, S_5, S_6 with $S_4 \subset S_5 \subset S_6$ are associated to treasury with levels LOW, MEDIUM and HIGH respectively;
- S_7, S_8, S_9 with $S_7 \subset S_8 \subset S_9$ are associated to market with level LOW, MEDIUM and HIGH respectively;

Then, during the Join procedure, users get keys according to their rights:

- A right for finance with security level LOW is associated with set S_1. A user U joining the system with this right receives sk_1;
- A right for market with security level MEDIUM is associated with sets $S_7 \subset S_8$. A user U joining the system with this right receives both sk_7 and sk_8.
- A right with treasury with security level HIGH is associated with sets $S_4 \subset S_5 \subset S_6$. A user U joining the system with this right obtains sk_4, sk_5, and sk_6.

3 CoverCrypt

Let $\mathbb{G}$ be a group of prime order q, with a generator g, in which the Computational Diffie-Hellman problem is hard. We describe below the encryption mechanism that can target specific rights for decryption, and traceability of dishonest users:

- Setup$((S_i)_i)$: it generates the master public key mpk and the master secret key msk as follows:
  - it samples random $u, v, s \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets

$$U \leftarrow g^u \qquad V \leftarrow g^v \qquad H \leftarrow g^s$$

  - For each set $S_i \in \mathcal{S}$, where $\mathcal{S} = (S_i)_i$, it chooses a random $x_i \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets $H_i \leftarrow H^{x_i}$.

Let msk $\leftarrow (u, v, s, (x_i)_i)$ and mpk $\leftarrow (\mathbb{G}, g, U, V, H, (H_i)_i)$. And we assume that mpk is known to everybody.

- Join(msk, j, A_j): it takes as input the master secret key msk, a user identifier j, and the set A_j of indices i so that user j belongs to S_i, and provides its secret key SK$_j$.

For the tracing, one first chooses random scalars (a_j, b_j) such that

$$a_j \cdot u + b_j \cdot v = s$$

Then SK$_j \leftarrow (a_j, b_j, (x_i)_{i \in A_j})$ is provided to user j.

- Enc(K, B): it takes as input a bitstring $K \in \{0,1\}^n$ to encrypt to all the users belonging to S_i, for $i \in B$, and outputs the encryption of K.
  - it samples a random $r \overset{\$}{\leftarrow} \mathbb{Z}_q$;
  - it sets $C \leftarrow U^r$ and $D \leftarrow V^r$;
  - for every $i \in B$, it generates $K_i \leftarrow H_i^r$.

The ciphertext thus consists of $(C, D, (E_i = \mathcal{H}(K_i) \oplus K)_{i \in B})$, where $\mathcal{H}$ is a hash function onto $\{0,1\}^n$.

- Dec(SK$_j, (C, D, (E_i = K_i \oplus K)_{i \in B})$): it takes as input a user's secret key and a ciphertext, it outputs the encrypted key K.
  - the user first chooses an index $i \in B \cap A_j$, in both its set of rights A_j and the rights B of the ciphertext, and then uses $x_i =$ sk$_i \in$ SK$_j$;
  - it can compute $K_i = (C^{a_j} D^{b_j})^{x_i}$, and extract $K = E_i \oplus \mathcal{H}(K_i)$.

One can note that

$$K_i = (C^{a_j} D^{b_j})^{x_i} = (g^{ura_j + vrb_j})^{x_i} = (g^{(ua_j + vb_j)r})^{x_i} = g^{srx_i}$$

4 Security Properties

The previous construction provides two security properties: the privacy of the encrypted key K and the traceability of users. More precisely:

4.0.1 Property 1: privacy

Any collusion of users does not learn more than what the users could get individually and then put in common.

The security game is the following one: an adversary $\mathcal{A}$ corrupts users and learns their secret keys SK_j for $j \in \mathcal{C}$ (the set of corrupted users), and asks for the encryption $K^{(b)}$ among $\{K^{(0)}, K^{(1)}\}$ for the set B. We say that the adversary wins the security game if it can guess b, while $B \cap (\cup_{j\in\mathcal{C}} A_j) = \emptyset$.

In the static corruption setting, we know the set $\mathcal{C}$ before the Setup, for which we are given a Diffie-Hellman instance $(g, X = g^x, Y = g^y)$:

- $U \leftarrow g^u$, $V \leftarrow g^v$, and $H \leftarrow g^s$, for random u, v, s;
- For $i \in \cup_{j\in\mathcal{C}} A_j$, choose a random x_i and set $H_i \leftarrow H^{x_i} = g^{sx_i}$;
- For $i \notin \cup_{j\in\mathcal{C}} A_j$, choose a random y_i and set $H_i \leftarrow X^{sy_i} = g^{sxy_i}$, which virtually sets $x_i \leftarrow xy_i$;

Join-queries can be generated as in the official procedure, as for corrupted users, x_i is known. However, when encrypting $K^{(b)}$ for the set B:

- $C \leftarrow Y^u = g^{yu}$, $D \leftarrow Y^v = g^{yv}$, which virtually sets $r \leftarrow y$;
- For $i \in B$, it generates a random $E_i \overset{\$}{\leftarrow} \{0,1\}^n$.

The adversary can only see the difference of the random E_i if it asks for $\mathcal{H}(K_i)$, for some $i \in B$, while $B \cap (\cup_{j\in\mathcal{C}} A_j) = \emptyset$: $K_i = H_i^r = g^{sxy_iy} = (g^{xy})^{sy_i}$. From all the queries to $\mathcal{H}$, in the random oracle model, and s and $(y_i)_i$, one can extract g^{xy}, which is the Diffie-Hellman value of (X, Y) in basis g.

4.0.2 Property 2: traceability

Without any collusion, a user alone can be traced with a black-box tracing procedure.

In order to trace a pirate decoder that contains a key (a, b): one generates a fake ciphertext with $C \leftarrow U^r \cdot g^{bs}$ and $D \leftarrow V^r \cdot g^{-as}$ for random $r, s \overset{\$}{\leftarrow} \mathbb{Z}_q$, and then $K_i \leftarrow H_i^r$.

User with key (a, b) will compute $C^a D^b = U^{ar} \cdot g^{abs} \cdot V^{br} \cdot g^{-abs} = (U^a \cdot V^b)^r = H^r$, and will eventually get the correct key K if it owns x_i.

Other users will compute $C^{a'} D^{b'} = U^{a'r} \cdot g^{a'bs} \cdot V^{b'r} \cdot g^{-ab's} = (U^{a'} \cdot V^{b'})^r \cdot g^{(a'b-ab')s} = H^r \cdot g^{(a'b-ab')s}$, which will unlikely be correct as $a'b - ab' \neq 0 \bmod q$.

4.0.3 Property 3: 1-collusion resistance

Any collusion of 2 or more users can build an anonymous key.

As we have:

$$a_1 u + b_1 v = s \qquad a_2 u + b_2 v = s$$

any convex combination, with $\alpha + \beta = 1$ leads to an anonymous key:

$$(\alpha a_1 + \beta a_2)u + (\alpha b_1 + \beta b_2)v = s$$

Indeed, we have

$$\alpha a_1 u + \alpha b_1 v + \beta a_2 u + \beta b_2 v = (\alpha + \beta)s$$

Hence, $(a = \alpha a_1 + \beta a_2, b = \alpha b_1 + \beta b_2)$ is an anonymous key that cannot be traced.

5 Updates

5.0.1 Adding Users and Rights

A new user joining the system will receive secret keys associated to its rights. These rights may evolve and the policy can be enriched over the time with more subsets S_i, and thus more keys sk_i.

5.0.2 Revocation

For revocation, one can use time-periods, and then a three-dimensional space, with $\mathsf{sk}_{t,i}$. When the time-period changes, users receives the keys $\mathsf{sk}_{t,i}$ associated to their right. The tracing part (a_j, b_j) does not need to be updated.

On the other hand, stored data does not need to be re-encrypted, but the key K must be re-encrypted under the new $\mathsf{sk}_{t,i}$. It can be done without any private information: but we need a slight modification with $\mathsf{Enc}(K, B)$ that now takes as input a key $K \in \mathbb{G}$, while $H_{t,i} = H^{x_{t,i}}$ depends on the time-period t

- it samples a random $r \overset{\$}{\leftarrow} \mathbb{Z}_q$;
- it sets $C \leftarrow U^r$ and $D \leftarrow V^r$;
- for every $i \in B$, it generates $K_{t,i} \leftarrow H_{t,i}^r$ and $E_{t,i} = K \times K_{t,i}$

The ciphertext of K is thus $(C, D, (E_{t,i})_i)$, and data is encrypted with the session key $\mathcal{H}(K) \in \{0,1\}^n$.

When updating the keys: $H_{t+1,i} \leftarrow H_{t,i} \cdot U^{\Delta_{t,i}} = H^{x_{t,i}+\Delta_{t,i} \cdot u/s}$. The ciphertext should be updated so that

$$E_{t+1,i} = K \times H_{t+1,i}^r = K \times H_{t,i}^r \cdot U^{r \cdot \Delta_{t,i}} = E_{t,i} \cdot C^{\Delta_{t,i}}$$

6 Comparison with GPSW06 (ABE)

In short, CoverCrypt uses simpler, older primitives, it is faster and the ciphertexts are more compact.

Side by side

	GPSW06	CoverCrypt
Protocol Family	Attributes Based Encryption (ABE)	Subset Cover
Primitives	Pairings over Elliptic Curves	El Gamal over Elliptic Curves (no pairings)
Security	~128 bits using Curve BLS12-381	~128 bits using Curve 25519 . The implementatation of X25519 follows RFC7748
Attacks	SeeAttacks §5 and §5.1 for attacks on pairings.	SeeAttacks §5
Encoding	Attributes are directly encoded	The combination of attributes determines the list of all possible partitions/sets. Partitions are then encoded. Assuming 2 axes Departments [HR, R&D] and Security [Low, High], there are up to 4 possible partitions: HR-Low, HR-High, R&D-Low, R&D-High
Ciphertexts header size (bytes)	$332 + \gamma * 52$ where γ is the number of attributes of the ciphertext (in most use cases, $\gamma = 2$)	$67 + \sigma * 33$ where σ is the number of partitions for the ciphertext (in most use cases, $\sigma = 1$)
Ciphertexts body size (bytes)	AES GCM: $C + 28$ where C is the clear text size	AES GCM: $C + 28$ where C is the clear text size

Attacks (and key sizes) Please check this INRIA reference document

7 Implementation choices

7.1 Curve25519

The curve used is the X25519 elliptic curve. The rust implementation is the `curve25519-dalek`. It is a library providing group operations on the Edwards and Montgomery forms of Curve25519, and on the prime-order Ristretto group. In particular, it implements Ristretto, which constructs a prime-order group from a non-prime-order Edwards curve. This provides the speed and safety benefits of Edwards curve arithmetic, without the pitfalls of cofactor-related abstraction mismatches. Curve25519 is `approved by the NIST` and is described in `RFC7748`. It offers 128 bits of security.

7.2 PRG

To generate random numbers, we use the `Rust implementation` of the HC-128 algorithm from the standard library. HC-128 offers 128 bits of security.

7.3 Hash function

As hash function, we use `the implementation` of the Sha256 algorithm from the standard library, which offers 128 bits of security. We use this hash function to generate a KDF using the HKDF (RFC5869) implementation from the `hkdf` crate.

7.4 DEM

As DEM, we use our implementation of the ISO/IEC 18033 standard. As symmetric scheme, we use AES implementation of `aes_gcm`. This implementation provides encryption and decryption functions that also manage authentification. We use an AES key length of 256 bits and a MAC length of 128 bits. The nonce length used is 96 bits.