Introduction to blockchain December 25, 2017

Shamir's secret sharing scheme

Evgeny Marshakov, Mikhail Burov

1 Task

Make an application with a GUI which

- allows uploading png image as an input,
- encrypts it with a unique key,
- splits the key for a threshold cryptography with a pre-defined parameters (k, n),
- as an output provides encrypted image and a set of key parts.

Prepare project's demo.

2 Theoretical part

2.1 Shamir's secret sharing

Suppose we want to split secret c in (k, n)- threshold scheme. Without loss of generality, we can assume that $c \in \mathbb{F}_p$ for some prime number p. The key idea of the construction presented below is that any polynomial of degree $k-1$ can be determined by its value in k points.

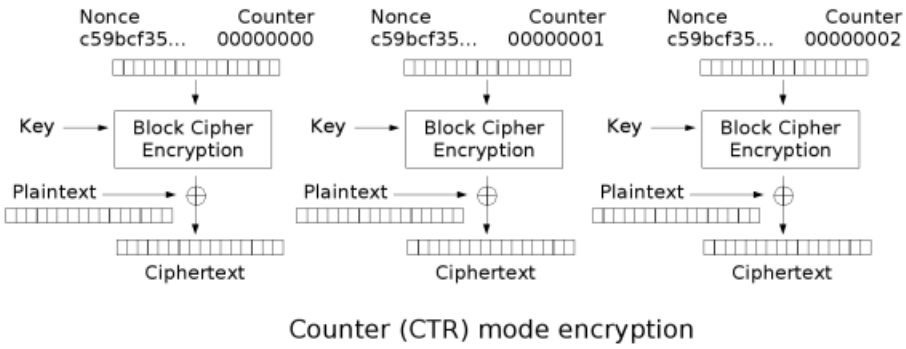
Let $a_1, \ldots, a_{k-1} \in \mathbb{F}_p$ be a randomly generated elements. So we can construct a polynomial in the following way

$$F_c(x) = c + a_1 x + \cdots + a_{k-1} x^{k-1} \in \mathbb{F}[x].$$

Now we can easily split the secret key s. Every participant of the scheme will obtain value of the polynomial $F_c(x)$ at some point $x \neq 0$ called *shadow* of the secret. For simplicity, we can share $F_c(i)$ for i-th participant $i = 1, \ldots, n$. Then k participants will be sufficient to recover secret key using, for example, Lagrange interpolating polynomial or system of linear equations w.r.t $c, a_1, \ldots, a_{k-1}$.

2.2 AES in CTR mode

For the ciphering we use Advanced Enryption Standard (AES) in counter mode with key of size 256 bits. Below we can see the scheme of this procedure.



Counter (CTR) mode encryption

So the cipher text of i-th block can be obtained by the following formula

$$C_i = P_i \oplus E_k(Ctr_i)$$

where P_i is the i-th block of a plaintext, $E_k(\cdot)$ - block ciphering function with key k.

3 Realization

All code was written on Java. For GUI we use JavaFX.

Instead of simple XOR operation of an image bitmap with the random sequence size of width*height*4 bytes (ARGB) as the secret key, we use 256-bit length key for AES in CTR mode without padding. It helps us to significantly reduce the secret key size and increase the performance. In good cryptographic practices, we didn't implement AES and decided to use standard encryption extension from the javax.crypto package.

Please, don't use it in real conditions. It's just a toy!

3.1 Algorithms

We assume that p and (k, n) are known for everybody.

Algorithm 1: Split the key

input : c - secret, (k, n) - threshold parameters;
output: $c_1, \ldots, c_n$ - shadows;
begin
  for $i = 1, i < k, i++$ **do**
    Randomly generate $a_i \in \mathbb{F}_p$;
  end
  $F_c(x) = c + a_1 x + \cdots + a_{k-1} x^{k-1} \in \mathbb{F}_p[x]$;
  for $i = 1, i < n + 1, i++$ **do**
    $c_i = F_c(i)$
  end
end

Algorithm 2: Combine the key

input : $c_{i_1}, \ldots, c_{i_k}$ - shadows;
output: c - secret
begin
  for $j = 1, j < k, j++$ **do**

$$L_{i_j} = \prod_{\substack{1 \le l \le k \\ l \ne j}} \left[-i_l \cdot (i_j - i_l)^{-1} \right] \mod p$$

  end
  $c = \sum_{j=1}^{k} c_{i_j} \cdot L_{i_j}$
end