Rust Crypto For XChain

Xuper-sdk-go vs rust-sgx for Crypto

	超级链	crate-rust-sgx
ecdsa	crypto/ecdsa: P256-SHA256-ANS1	ring::P256-SHA256-ASN1 ring::P256-SHA384-ASN1
hash	crypto/hmac crypto/sha512 crypto/sha256 "golang.org/x/crypto/ripemd160"	ring::{hmac,sha256,sha512} ripemd160
encode	self/base58 自己实现的	base58
bigint	math/bigint	num-bigint
rand	crypto/rand	rand
aes	crypto/aes (Rijndael 128, 192, 256)	ring
ecies	Kylom's implementation curve: P256	Done
sign	multi_sign, schnorr_ring_sign, schnorr_sign	**需要实现**
hdwallet/keychain	hdwallet/keychain	**需要实现**

ECC

ECDSA

Parameter	
CURVE	the elliptic curve field and equation used
G	elliptic curve base point, a point on the curve that generates a subgroup of large prime order n
n	integer order of *G*, means that n x G=O , where O is the identity element.
k	the private key (randomly selected)
P	the public key (calculated by elliptic curve)
M	the message to send

ECDSA签名：

$$
\begin{gathered}
P=(x_1,y_1)=k\times G \\
S=k^{-1}(Hash(M)+k*x_1)\ mod\ p \\
Signature=(x_1,S)
\end{gathered}
$$

ECDSA 验证签名

$$
\begin{gathered}
P^{'}=S^{-1}*Hash(M)\times G+S^{-1}*x_1\times P \\
=P
\end{gathered}
$$

- 证明

$$
\begin{gathered}
P^{'}=S^{-1}*Hash(M)\times G+S^{-1}*k\times G \\
=(S^{-1}*Hash(M)+S^{-1}*k)\times G \\
=(Hash(M)+x_1)*S^{-1}\times G \\
=(Hash(M)+x_1)*(k^{-1}(Hash(M)+k))^{-1}\times G \\
=(Hash(M)+x_1)*k*(Hash(M)+k)^{-1}\times G \\
=k\times G \\
=(x_1,y_1)
\end{gathered}
$$

ECIES算法

为了向Bob发送ECIES加密信息，Alice需要以下信息：

- 密码学套件（ KDF，MAC，对称加密E）
- 椭圆曲线(p, a, b, G, n, h)

- Bob的公钥:

$$K_b, K_b = k_b G, k_b \in [1, n-1]$$

- 共享信息

$$S_1, S_2$$

- 无穷远点O

加密

Alice使用Bob的公钥加密消息m：

$$\begin{gathered}
For\ random\ r \in [1, n-1], calculate\ R = rG \\
derive\ shared\ secret: S = P_x, where\ P = P(P_x, P_y) = rK_b, P \neq O \\
derive\ K_E || K_M = KDF(S || S_1) \\
encrypt\ message\ m: c = E(k_E; m) \\
calculate\ MAC: d = MAC(k_M; c || S_2) \\
output: R || c || d
\end{gathered}$$

解密

Bob解密密文 R||c||d的步骤如下：

$$\begin{gathered}
derive\ shared\ secret: S = P_x, P = P(P_x, P_y) = k_B R \\
derive\ K_E || K_M = KDF(S || S_1) \\
verify\ MAC: d == MAC(k_M; c || S_2) \\
decrypt: m = E^{-1}(k_E; c)
\end{gathered}$$

证明过程

we need ensure S is really shared by Alice and Bob:

$$P = K_B r = k_B R$$

Refer

1. https://en.wikipedia.org/wiki/Integrated_Encryption_Scheme

有限域运算

密码学主要研究2种有限域： 大素数域以及特征为2的有限域。

表示

有限域：

$$GF(p^n)$$

这里专门针对p=2为特征的多项式进行计算。

多项式基表示法

$$A = \alpha_{m-1}\beta^{m-1} + \ldots + \alpha_1\beta + \alpha_0$$

正规基表示法

$$A = \sum_{i=0}^{m-1} \alpha_i \beta^{2^i}, where\ \alpha_i \in GF(2), i \le i \le m-1$$

运算

模加法

1. 点加法

$$\lambda = \begin{cases} x = & (y_p - y_q)/(x_p - x_q)\ mod\ p, P! = Q \\ y = & (3x_p^2 + a)/y_p\ mod\ p, P = Q \end{cases}$$

$$\begin{aligned} x_r &= (\lambda^2 - x_p - x_q)\ mod\ p \\ y_r &= (\lambda(x_p - x_r) - y_p)\ mod\ p \end{aligned}$$

转成Jacobian格式(转换方法： (x, y) -> (x',y', 1), (x', y', z) -> (x= x' /z²,y= y' /z³))之后，计算流程如下（推导过过程省略）：

$$\begin{aligned} u1 &= x1 \cdot z2^2 \\ u2 &= x2 \cdot z1^2 \\ s1 &= y1 \cdot z2^3 \\ s2 &= y2 \cdot z1^3 \\ h &= u2 - u1 \\ r &= s2 - s1 \\ x3 &= r^2 - h^2 - 2 \cdot u1 \cdot h^2 \\ y3 &= r \cdot (u1 \cdot h^2 - x3) - s1 \cdot h^3 \\ z3 &= z1 \cdot z2 \cdot h \end{aligned}$$

2. 二倍运算

相当于p=q的场景，代入上面计算。

模乘法

Montgemory模乘

模仿快速幂，然后利用点加进行计算。例如Golang的实现ScalarMut.

例如快速幂的思想如下： a^b % p伪代码如下：

```
long long Montgomery()
{
    long long int res = 1;
    while(exp_1)
    {
        if (exp_1&1)//如果为奇数
            res = (res*base) % mod;
        exp_1 >>= 1;//指数对半
        base = (base*base) % mod;//底数平方
    }
    return res;
}
```

原理如下：

$$a^b = a^{\sum_{i=0}^{k} 2^{k_i}}, k_i \in \{0, 1\}$$

将b分解为二进制之后，从k_0开始计算， 如果遇到当前二进制**位**为偶数:

$$\begin{aligned} s' &= s \\ base &= base * base \end{aligned}$$

如果b为奇数：

$$\begin{aligned} s' &= s * base \\ base &= base * base \end{aligned}$$

那么对于椭圆曲线上的点乘法，可以采用类似的思路，分解k为二进制，然后使用二倍运算计算base，点加计算s。

模逆

模逆基(a^{-1} mod p)本上采用的是扩展欧几里算法。如果p是素数，那么可以用Femat's little theorem:

$$\begin{aligned} a^{p-1} &= 1 \, mod \, p \\ a^{p-2} &= a^{-1} \, mod \, p \end{aligned}$$

然后利用上面的点乘解决。

要计算a^{-1} mod p, 可以写成求最小的正整数x, y, 使得： ax + by = 1。 注意p重新命名为b，明显gcd(a, b) = 1.

充分利用gcd(a , b) = gcd(b , a % b), 最终求得x=d，伪代码如下:

```
Extend-Ecuclid(a,b):
  If b == 0:
    Then return(a,1,0)
  (d1,x1,y1) = Extend-Ecuclid(b,a mod b)
  (d,x,y) = (d,y1,x1-[a / b] * y1)
  Return (d,x,y)
```