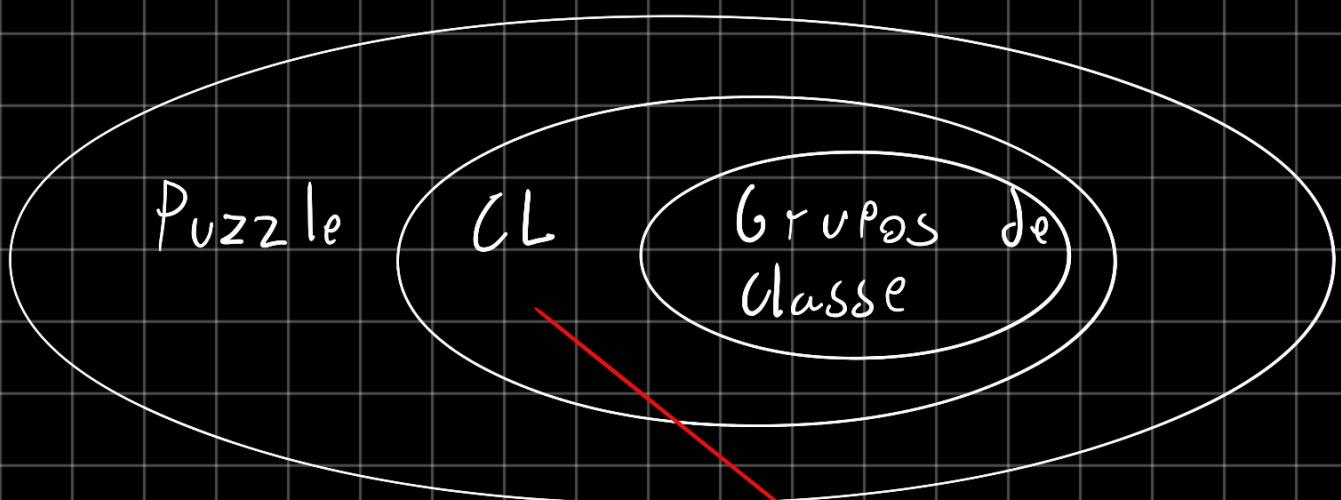


A²L : Mapa de conceitos



- Puzzles usam CL
- CL usa grupos de classe
- Grupos de classe quadráticos

→ Esquema de criptografia homomórfica (linear)

//

Grupos de classe quadráticos:

$$(f(x, y) = ax^2 + bxy + cy^2 \quad \forall a, b, c \in \mathbb{R}^*)$$

→ Isto é uma forma quadrática bimária.

$$(f = (a, b, c);$$

→ Uma forma, os objetos que trabalharemos.

Discriminante de Uma Forma:

$a \equiv b \pmod{m} \rightarrow$ Dividindo "a" por "b" o resto é "m".

$\Delta(f) = b^2 - 4ac \rightarrow$ Forma do discriminante.

$(a, b, c) \in \mathbb{Z} \rightarrow b \equiv \Delta(f) \pmod{2}$

\hookrightarrow Ou ambos b e $\Delta(f)$ são pares ou ímpares.

• Um discriminante fundamental define quais são ideais para construir estruturas algébricas sólidas, como os campos quadráticos e grupos de classe.

Temos dois casos fundamentais:

1) $\Delta \equiv 1 \pmod{4}$: Nenhum número quadrado divide Δ .

2) $\Delta \equiv 0 \pmod{4}$: Só é fundamental se $\frac{\Delta}{4} \equiv 0, 3 \pmod{4}$, $\frac{\Delta}{4}$ for "square-free".

\hookrightarrow Discriminantes fundamentais são exatamente aqueles que são discriminantes de campos quadráticos.

Redução de Gauss:

Considere uma caixa-preta, apenas funciona.

Ao multiplicar duas formas quadráticas, os coeficientes crescem absurdamente, sem reduzir a memória do computador acabaria em segundos.

↳ A redução de Gauss pega uma forma gigante e encontra uma forma equivalente com os menores coeficientes possíveis.

↳ As funções "mpow" e "gmul" da PARI (que vamos usar) fazem isso implicitamente.

A forma $f, g-q$ e os coeficientes (a, b, c) :

• O problema do logaritmo discreto:

Existem 2 geradores conceituais, dentro desse mesmo grupo de ordem " q " (compatível com Bitcoin), existem dois tipos de elementos com papéis opostos:

→ $g-q$ (o gerador difícil): É uma forma quadrática criada a partir dos coeficientes (a, b, c) que estão hardcoded na implementação do Tairi.

$$X = g^y \pmod{p}$$

O problema do log discreto se define que precisamos encontrar y e temos X, g, p , sendo p um primo, e g é o gerador do grupo definido por p .

O g é tal que elevando-o de 1 a $p-1$, ele gera todos os elementos do grupo $(g^1, g^2, \dots, g^{p-1}) \pmod{p}$.

$$X = 27893^y \pmod{61981}$$

Onde y é o segredo e X é o resultado público (chave pública).

O problema do log discreto pode ser definido da seguinte forma:

"Se eu te der a posição de um ponto na horizontal (X), você consegue me dizer a altura dele (y) sem ter que testar um por um?"

↳ Não. Como não existe uma linha ou curva contínua conectando os pontos (eles são discretos, ou seja, separados), não há como usar cálculos ou cálculos geométricos para achar y .

↳ O gerador " g " é um número escolhido que garante que ao elevar a potências sucessivas ele "pula" por todos os lugares possíveis de forma caótica antes de começar a repetir.

• Pequeno Teorema de Fermat

e é uma tarefa computacionalmente brutal.

(a, b, c) estão chumbados no código pois $g^a \cdot g^b \equiv g^c \pmod{p}$.
Ser compatível com BTC.

Voltando ao gerador difícil ($g^x \pmod q$):

Gerado pelas coef. (a, b, c) hardcoded.

Log discreto difícil: $g^x \pmod q$.

É usado para gerar as chaves (pk, sk).

$$\hookrightarrow \begin{array}{l} sk = x \in \mathbb{Z}_q \\ pk = g^x \pmod q \end{array}$$

Usados no subgrupo de log discreto fácil.

→ O gerador fácil (implícito):

O log discreto mele é fácil.

$$f_m = (a = g^2, b = L \cdot g, c = (L^2 - \Delta_k)/4)$$

Onde L é o plaintext/secreto (m): $L = (m^{-1}) \pmod q$

\hookrightarrow Dado f_m basta olhar para o segundo coeficiente e dividir por g para achar o secreto (L).

O correto seria pegar um gerador fácil (f) e elevar a " m ", entretanto o Taiti já sabe a forma final disso, então ele já monta f_m pronto conforme visto

acima.

→ A mágica da encriptação: Esquema CL

$$C_1 = g \cdot q^r, \text{ onde } r \in \mathbb{Z}_q$$

$$PK = g \cdot q^{sk}$$

$$C_2 = \underbrace{PK^r}_{\text{Máscara vindo}} \cdot \underbrace{f_m}_{\text{Mensagem no}}$$

Máscara vindo
do gerador difícil

Mensagem no
formato fácil (seria f^m)

- O $g \cdot q$ usado garante que a máscara PK^r seja indecifrável (problema do log discreto).
- C_1 é puramente do grupo difícil e é uma pista da máscara usada para decifrar.

→ A mágica da decriptação:

C_1 é a pista do desbloqueio da criptografia.

O receptor não sabe "r" da máscara (pk^r), o C_1 permite que o receptor receba a máscara usando a chave privada (sk).

$$pk = g \cdot q^{sk} \quad \text{e} \quad C_1 = g \cdot q^r$$

$$C_1^{sk} = (g \cdot q^r)^{sk} = (g \cdot q^{sk})^r = pk^r = \text{Máscara}$$

↳ Dessa forma, tendo a máscara, é possível limpar C_2 do grupo difícil e acessar o segredo pelo grupo fácil.

- Essa propriedade permite remover a máscara mesmos com as impróprias randomizações que o A2L exige, a lógica é análoga.

$$C_1'' = (g \cdot q^r)^{\beta \cdot \pi} ; \quad C_2'' = pk^{r \cdot \beta \cdot \pi} \cdot f_m^{\beta \cdot \pi}$$

$$(C_1'')^{sk} = (g \cdot q^{sk})^{r \cdot \beta \cdot \pi} = pk^{r \cdot \beta \cdot \pi}$$

→ O efeito da randomização no f_m e no decifrador:

$$f_m(L) = \left(q^2, L \cdot q, \frac{L - \Delta_K}{q} \right)$$

$$f_m(L)^K = f_m(L \cdot K)$$

Propriedade homomórfica aditiva.

$$f_m^K = \left(q^2, L \cdot q \cdot k, \frac{L^2 - \Delta_K}{q} \right)$$

O segredo final lido será "randomizado" por K :

$$\hookrightarrow \frac{L \cdot K \cdot q}{q} = L \cdot K = \alpha'$$

$$\alpha = \frac{\alpha'}{K} = \frac{L \cdot K}{K} = L$$

\hookrightarrow Processo inverso calculado pelo receptor (P_r) randomizado.

//

Sobre Adaptor Signatures:

O segredo foi tratado na seção anterior como " m ", ou " L ", mas aqui será tratado como " α ", que é como o paper trata.

- Mundo criptográfico off-chain: Resolve-se puzzles, somam-se formas quadráticas e manipula-se o segredo " α " no "escuro" (criptado).

O espaço de mensagens é \mathbb{Z}_q .

- Mundo blockchain (curva elíptica): Onde o Bitcoin valida assinaturas. As chaves privadas (sk) e os moncos "k" são escalares que vivem no corpo \mathbb{Z}_q , onde " q " é a ordem da curva "SECP 256KL".

* Ordem e módulo funcionam como sinônimos praticamente para definir o tamanho do espaço.

↳ problema da ordem " q " no CL e no Bitcoin:

Se $q_{CL} = 60 \neq q_{BRC} = 12$:

$$A = g^d \quad \text{e} \quad d = 13$$

Na blockchain: $d = 13 = 1 \pmod{12}$

No CL/Puzzle: $d = 13$; Não extrapola $q = 60$.

→ O " d " precisa ser compatível entre o puzzle e a assinatura na Blockchain, logo o \mathbb{Z}_q de CL precisa coincidir com a ECC do Bitcoin.

→ A forma de uma adaptor Signature:

- Uma assinatura "quebrada", ou pré-assinatura.

○ ⇒ Assinatura válida, vai para a blockchain.

$\hat{\sigma} \Rightarrow$ Assinatura válida, faltando somar o segredo d .

Schnorr:

$$R = g^K, \text{ nonce } K \quad \curvearrowright \text{ Transações.}$$

Desafio: $e = \text{Hash}(R, pk, tx)$
↳ Mistura tudo em um hash.

$s = k + e \cdot sk \Rightarrow$ Assinando com chave privada.

No adaptor Signature:

$$A = g^d \quad e \quad R' = R \cdot A \Rightarrow R' = g^K \cdot g^d = g^{k+d}$$

$$e' = \text{Hash}(R', pk, tx)$$

$$\boxed{s = k + e' \cdot sk} \rightarrow \text{Assinatura "quebrada"}$$

• O nonce usado foi $k+d$, a assinatura mitou no lugar "errado".

↳ Entretanto para essa assinatura funcionar nonce somado precisa ser $k+d$, só somamos k .

↳ Ao somar d , a assinatura se torna válida.

$$\left. \begin{array}{l} S = \hat{S} + \alpha \\ \sigma = \hat{\sigma} + \alpha \end{array} \right\} \quad \left. \begin{array}{l} S = (k + e \cdot sk) + \alpha \\ \sigma = " " + \alpha \end{array} \right.$$

→ Verificação na blockchain:

$$g^S = R \cdot p_k^e$$

O Bitcoin recebe R (o ponto "mentiroso"), p_k , " e " (o hash da transação) e o S final.

$$g^{(k+e \cdot sk)} = g^k \cdot g^{e \cdot sk}$$

$$R = g^k \quad e \quad p_k = g^{sk}$$

$$g^{(k+e \cdot sk)} = g^k \cdot (g^{sk})^e$$

$$\therefore \boxed{g^{(k+e \cdot sk)} = R \cdot p_k^e} \Rightarrow g^S = R \cdot p_k^e$$

Para a adaptor signature é análogo:

$$\boxed{g^S = R' \cdot p_k^{e'}}$$

Se isso for verdadeiro, a assinatura é válida.

~ // ~

Apêndice: