

# Análise de Funções Hash

1

## 1. Funções de Hash

Funções de hash são funções que mapeiam  
arquivos arbitrários (tipicamente strings não-  
limitadas de bits) em strings de bits de tamanho  
fixo.

$$h: \{0,1\}^* \rightarrow \{0,1\}^t$$

Em criptografia estas funções têm inúmeras  
aplicações. Isto porque uma boa função de hash  
verifica uma conjunto de propriedades.

- São implementáveis de forma muito eficiente. Mesmo que  $s$  tenha muitos mega bytes  
de tamanho,  $\underline{h(s)}$  calcula-se em milisegundos
- São nov, inventivas:
  - a) Se for  $x = h(s)$ , a partir do conhecimento de  $x$ , não é possível calcular  $s$
  - b) Conhecendo  $x, s$  não é possível calcular  $s' \neq s$  tal que  $\underline{h(s')} = \cancel{h(x)}$
  - c) Não é possível calcular um par  $s, s'$  com  $s \neq s'$  tal que  $\underline{h(s)} = h(s')$

- ② As funções de Hash: mais algoritmos standard
- O mais antigo é conhecido por MD5 e é um hash com "outputs" de 128 bits
  - SHA-1 foi (e ainda é) um hash de 160 bits que faz parte do horário FIPS 186
  - foi substituído pelo SHA-2 (variantes sha256 e sha512), com comprimentos de 256 e 512 bits
  - Recentemente a FIPS aceitou como novo o SHA-3 o algoritmo Keccak (horário FIPS 202) que usa tamanhos de 24, 256, 384, 512 bits.  
Estes são tamanhos standard, mas o princípio Keccak pode ser configurado para qualquer tamanho.

### Message Authentication Codes (HMAC)

Pode usar funções de hash com chaves variadas construções são possíveis mas a mais comum é designada por HMAC

Veja página da Wikipédia para uma implementação Python com "hashlib".

## 2 - Derivar e gerir chaves

(3)

Uma das aplicações mais importantes das funções de hash é a geração de chaves, nomeadamente a derivação de chaves de grande entropia a partir de "passwords" de pequena entropia.

As passwords entendidas por humanos são de pequena entropia e facilmente podem ser atacadas por força bruta. Mesmo com chaves "razoáveis" atacar por hardware usando GPU's (muito mais rápidas que os normais CPU's) facilmente têm segundos para tentar todas as ~~possibilidades~~ possibilidades. Para resolver este problema usam-se funções de derivação.

~~PKDF~~: password  $\xrightarrow{\text{derivada}}$  chave de grande entropia

de tal modo que correr a função uma só vez usa recursos razoáveis, mas tentar usar esta função num ataque por força bruta é inatratável

- em termos de tempo CPU (1 core) -

mas não GPU's multi-core

~~algé~~ algoritmo PBKDF2

("password based key derivation function")

- em termos de memória: SCRYPT (instalar) e ARGON2

④ Une kOF time-hard memorise une  
une "salt" comme à "know"保管する  
pour une puissance de Hash aux niveaux élevés.  
de vez en (/ > 100 000)

Une kOF memory-hard via chiffrements  
semblants mais avec grande taille de  
mémoire.

### 3 - Primitivas de cifras simétricas

(5)

Cifras são primitivas criptográficas cujo objetivo é manter suas mensagens confidenciais.

cifra : Mensagem  $\rightarrow$  Criptograma

Cifre Inversa : Criptograma  $\rightarrow$  Mensagem

A cifre é segura quando:

- Só quem conhece a cifra inversa e o criptograma, conseguem recuperar a mensagem
- Mesmo conhecendo várias instâncias ( $m, \text{crypt}$ ) de pares (mensagem, criptograma) de anteriores operações de cifrar e decifrar, não procura de um novo criptograma (que não tenha ocorrido anteriormente) é impossível recorrer à mensagem que lhe deu origem, (sem conhecer "cifra inversa", obviamente)

Nas cifras modernas o conhecimento sobre a cifre e a cifra inversa é dividido por duas partes:

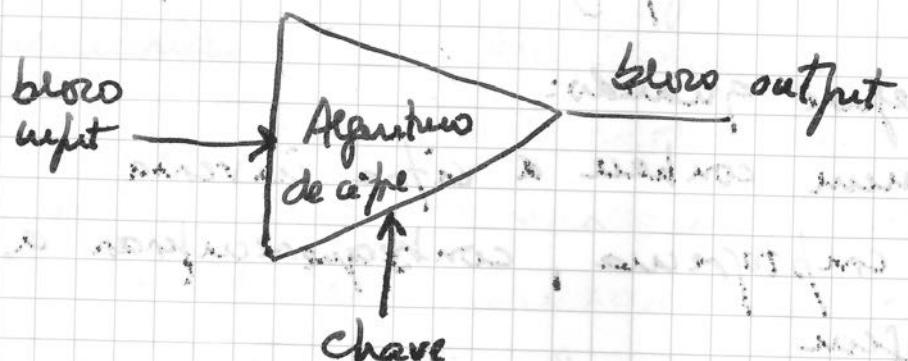
- o algoritmo (que é publicamente conhecido)
- a chave (que é secreta)

Quando a mesma chave é usada em ambas as operações, a cifra é simétrica.

6

Classificação da simétrica.

Um algoritmo de uma cifra simétrica por bloco atua sobre blocos de tamanho fixo e produz uma criptografia com o mesmo tamanho do "input".



$$|\text{chave}| \geq |\text{input}| = |\text{output}|$$

As mensagens têm tamanho aleatório; por isso é necessário partilhas em blocos mensagens.



bloco 1      bloco 2      ...      bloco N      padding

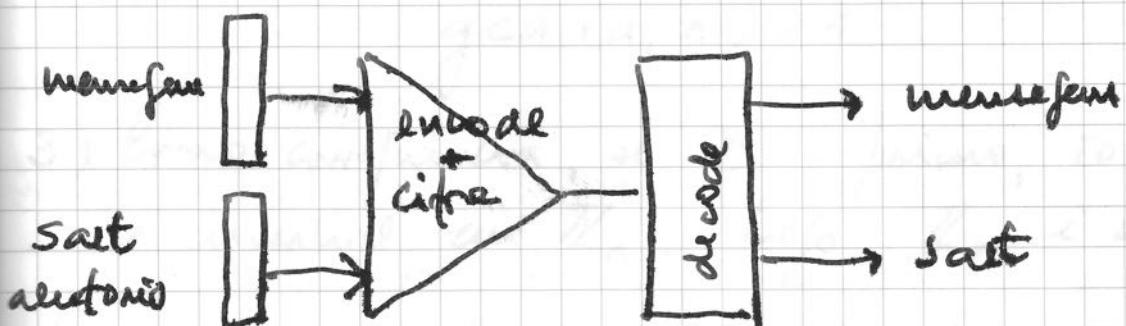
A forma acima é o modo de cifraria  
designado por padrão das cifras.

(ver figura 8.4 dos apontamentos) pag 252  
(ver figura 8.6 para GCM)

Parecendo os moldes, o uso de uma cifre simétrica moderna impõe o uso de um mecanismo de autenticação que garante a autenticidade do criptograma quando este é enviado por um qualquer mecanismo de comunicação.

Naturalmente recorre-se a um HMAC que usa a mesma chave que a ~~foi~~ usada na operação de cifra (ou seja derivado).

Finalmente pode-se recorrer a mecanismos de "randomization" (introdução de aleatoriedade) em que se mistura a mensagem com ~~com~~ uma parte aleatória que é suscetível de ser removida após a operação de decifrar



A vantagem deste mecanismo é o facto de tornar a cifre não-determinística (isto é, cifrar a mesma mensagem duas vez produz dois criptogramas diferentes) tornando-a muito mais difícil de atacar.

# Aula 1 - Inteiros, Inteiros Modulares, Criptostemas<sup>9</sup>

## RSA e ElGamal

### Inteiros Modulares

- 1) Em  $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$  define-se operações de soma e multiplicação da seguinte forma:
  - a)  $z = a + b$  em  $\mathbb{Z}_n$  se e só se  $z - (a+b)$  é um múltiplo de  $n$  em  $\mathbb{Z}$
  - b)  $z = a \times b$  em  $\mathbb{Z}_n$  se, e só se, nos inteiros  $(z - a \times b)$  é um múltiplo de  $n$ .
- 2)  $\mathbb{Z}_n$  é um anel com estas operações de soma e multiplicação.  
Os elementos de  $\mathbb{Z}_n$  invertíveis são aqueles que nos inteiros verificam
$$\gcd(a, n) = 1$$
- 3) Como consequência, se  $n$  é primo, todo  $a \neq 0$  é invertível em  $\mathbb{Z}_n$ . Logo  $\mathbb{Z}_n$  é um corpo.
- 4)  $\varphi(n)$  - a função Função Phi - dá o número de inteiros positivos, menores do que  $n$ , que são primos relativamente a  $n$ .

10.

5. Se  $n$  é primo, então  $\varphi(n) = n - 1$

$\varphi(n \times m) = \varphi(n) \times \varphi(m)$  se e só se

$\gcd(n, m) = 1$ , e  $\frac{1}{n} \in \mathbb{Z}_m$  são maiores do

que 1

6) Teorema de Euler e Variantes

a) Em  $\mathbb{Z}_n$ , para todo  $a \neq 0$

$$a^{\varphi(n)} = 1 \quad \text{em } \mathbb{Z}_n, \text{ se for}$$

$$k = i + j \pmod{\varphi(n)}$$

b) A igualdade em  $\mathbb{Z}_n$

$\varphi(n)$

$$a^k = 1$$

verifica-se sempre que uma das seguintes condições é válida

i)  $n$  é primo e  $a \neq 0$  (Teorema de Fermat)

ii)  $n$  é qualquer e  $a$  é inversível

(Teorema de Euler)

iii)  $n = p \times q$ , com  $p$  e  $q$  primos e  $a \neq 0$

(Teorema RSA)

c) Se  $i \equiv 0 \pmod{\varphi(n)}$  e todas as condições anteriores são válidas, então  $a^i = 1$

Ensas conclusões se definem dois condições: b<sub>i</sub>, ..., b<sub>iii</sub>

i) válida,

$$a^i \cdot a^j = a^k \quad \underline{\text{se } i+j \leq k}$$

$$k = i + j \bmod q(n)$$

$$a^{i \times j} = a \bmod n \quad \underline{\text{se } i \times j \leq 1}$$

$$i \times j = 1 \bmod q(n)$$

7). Grupo multiplicativo de  $\mathbb{Z}_n$

$\mathbb{Z}_n^*$  é o conjunto dos elementos de  $\mathbb{Z}_n$

que são invertíveis

i) formam um grupo multiplicativo

ii) Tem  $q(n)$  elementos

iii) Se  $n$  é um primo: ou:  $n = p \times f$  com  $p$

primo, existe um elemento de  $\mathbb{Z}_n^*$

que gera o multiplicativamente todo o

grupo

Isto é, para todo  $a \in \mathbb{Z}_n^*$  existe um índice  
 $0 \leq i < q(n)$  tal que  
 $a = g^i$  em  $\mathbb{Z}_n$

### 0. cálculo de exponentiação

$$x \mapsto g^x \bmod n$$

é executado por um algoritmo eficiente, mesmo para valores elevados de  $x, g, n$ . O algoritmo é

1. Calcular uma tabela de quadrados

$$g_i = g^{2^i} \bmod n$$

- calculados sequencialmente como

$$g_0 \leftarrow g$$

$$g_{i+1} \leftarrow g_i * g_i \bmod n$$

$i=0 \dots \text{tamanhos de } \varphi(n)$

2 - Representar o expoente  $x$  em notação binária

$$x = \sum_{i=0}^{\ell} b_i 2^i$$

$$\text{e } x = \sum_{i=0}^{\ell} b_i 2^i \quad b_i \in \{0, 1\}$$

3 - Calcular

$$g^x \bmod n = \prod_{b_i=1}^{\ell} g_i \bmod n$$

Notas: Se  $g, n$  forem constantes (para vários  $x$ )

o passo 1) é feito só uma vez e executa

$\leq |\varphi(n)|$  multiplicações.

O passo 3 tem também  $\leq |\varphi(n)|$  multiplicações.

Portanto o algoritmo é linear, com o número de

bits de  $\varphi(n)$

Infelizmente, o algoritmo inverso (DLP, algoritmo discreto)

$$\beta \mapsto x : \beta = g^x \text{ mod } n$$

tem complexidade que é linear com o maior primo que divide  $\varphi(n)$ .

Nota : Notar que não é limitado pelo tamanho do primo mas sim pelo próprio primo.

Muitas aplicações criptográficas usam valores de  $n$  da ordem de  $2^{1024}$ , e valores para o maior primo que divide  $\varphi(n)$  de ordem de  $2^{160}$ .

Assim a complexidade da exponenciação é da ordem de  $1024$  (tamanho de  $\varphi(n)$ ) mas o algoritmo inverso (DLP) tem complexidade da ordem de  $2^{160}$ .

Técnicas de Chave Pública da família Diffie-Hellman.

- ① Escolher um  $n$  primo, tal que  $n-1$  tem um divisor primo grande ( $\approx 2^{160}$ ) e  $n$  é de ordem de pelo menos  $\geq 1024$

② Gerar aleatoriamente um gerador  
 $g$  do grupo cíclico  $\mathbb{Z}_n^*$

③ Gerar aleatoriamente um segredo  $1 < s < n-1$   
 (a chave privada). Calcular a chave pública

$$\beta = g^s \text{ mod } n$$

que funciona como chave pública

### Algumas técnicas criptográficas da família DH

#### A) Protocolo de acordo de chaves DH

Dois agentes, A e B, cada um com um segredo  $a, b < n-1$ ,

enviam  $g^a$  e  $g^b$   $A \xrightarrow{g^a} B$   
 ao outro interlocutor  $\xleftarrow{g^b}$

Assim, para a ~~conhecer~~ conhecer sua chave

$$k = g^{ab}$$

O agente A recebe  $g^b$  e calcula  $k = (g^b)^a$

O agente B recebe  $g^a$  e calcula  $k = (g^a)^b$

Vai terceiro agente C conhecer  $g, g^a$  e  $g^b$

e não consegue saber se pode determinar  $k$

Tem de resolver, sem conhecer  $a$  ou  $b$ , o

seguinte problema

(15)

CDH

$$g, g^a, g^b \rightarrow g^{ab}$$

Este problema é conhecido por "computational DH problem" ou CDHP

Alternativamente tem de resolver

DLP

$$g, g^a \rightarrow a$$

Este é o novo-problema inverso, designado por "discrete logarithm problem"

Obviamente que CDHP se reduz a DLP: Se conseguimos resolver DLP determina-se  $a \in \mathbb{Z}$  e, com  $g^b$ , calcula-se  $g^{ab} = (g^b)^a$ .

A segurança criptográfica das técnicas DH baseia-se na anunção de que tanto CDHP como DLP são impraticáveis para parâmetros suficientemente grandes.

(B)

KEM - DEM

ElGamal

1 - Gerar um segredo "de senão"  $R < n-1$

2 - Calcular uma redundância

$$r = g^R \text{ mod } n$$

torne publica a redundância

KEM

3 - Calcular uma chave de sessão

$$k = \beta^R$$

e usá-la numa técnica de chave privada (e.g. uma cifra simétrica criando uma mensagem)

W - kere

Conhece a redundância  $\gamma$  e a chave secreta  $s$ . Representa a chave de sessão como

$$k \leftarrow \gamma^s$$

Not: ~~Tau = R + S~~,  $\gamma = g^R$  e  $\beta = g^S$

Calcular-se em kere,

$$k = \beta^R = (g^S)^R$$

em un-kere calcula-se

$$k = \gamma^S = (g^R)^S$$

### B C) Assinatura digital DSA

- Seja  $q$  o maior divisor primo de  $n-1$
- $H$  uma função de hash que gera valores no intervalo  $\mathbb{Z}$  contido em  $0..q-1$
- $m$  é a mensagem a assinar

#### Assinatura

- 1) gerar / pegar / pegar o par de chaves  $(k, B, Z, q-1)$  aleatoriamente

- Escolher um g que seja gerador do grupo aditivo de ordem q. Isto é:

$$g^q \equiv 1 \pmod{p}$$

- Chave privada é um sinalo  $1 < s < q$   
chave pública é  $\beta = g^s \pmod{p}$

Assinatura conhece a mensagem  $m$  e a chave privada  $s$

- 1) Gerar um numero aleatório  $R < q$  e calcular a respectiva redundância

$$r = (g^R \pmod{p}) \pmod{q}$$

Se  $r=0$  refeitar o processo com um novo  $R$  aleatório

- 2) Calcular

$$\sigma = R^{-1} (H(m) + sr) \pmod{q}$$

caso  $\sigma=0$  refeitar todo o processo com um novo  $R$  aleatório

- 3) A assinatura é o par

$$(r, \sigma)$$

Nota: O tamanho da assinatura é apena ximamente o dobro do tamanho de  $q$  (i.e. tipicamente 320 bits).  
Este é um tamanho muito menor que o tamanho de  $n$ , usado no RSA.  
Todas as operações de assinatura são feitas módulo  $q$ , portanto a 160 bits tipicamente.

Verificar a assinatura: conhece  $(r, \sigma)$ , em o.a.  
chave pública  $\beta$

1) Calcular expoentes

$$u = H(m) \cdot \sigma^{-1} \pmod{q}$$

$$v = r \cdot \sigma^{-1} \pmod{q}$$

2) Calcular

$$\lambda = g^u \beta^v \pmod{p}$$

3) Aceitar a assinatura se

$$\lambda \equiv r \pmod{q}$$

Correção

Como  $g^q \equiv 1 \pmod{p}$ , verifica-se sempre:

$$x \equiv y \pmod{q} \text{ implica } g^x \equiv g^y \pmod{p}$$

Temos

$$\begin{aligned} \lambda &\equiv g^u \beta^v \pmod{p} \\ &= g^u g^{sv} \pmod{p} = g^{(u+sv)} \pmod{p} \end{aligned}$$

$$\text{porque } \beta = g^s \text{ então } \beta^v = g^{sv}$$

Temos

$$u + sv = (H(m) + sr) \cdot \sigma^{-1} \pmod{q}$$

como  $\sigma^{-1} \equiv r^{-1} \pmod{q}$ , logo

$$(u + sv) \equiv R \pmod{q}$$

Logo  $g^{(u+sv)} = g^R \pmod{p}$

↑ porque  $r = (g^R \pmod{p}) \pmod{q}$

Agora  $\boxed{\lambda = r \pmod{q}}$

## Aula 2 - Polinômios

Frequentemente é necessário enriquecer, nas aplicações criptográficas, as propriedades algébricas dos inteiros.

A estrutura seguinte é a dos polinômios, não só porque preservam muitas das propriedades dos inteiros como têm uma longa tradição de uso em Telecomunicações, nomeadamente nos códigos corretivos de erros de transmissão.

Genéricamente para um qualquer anel  $R$  representamos por  $R[x]$ ,  $R[x, y] \dots$  os anéis dos polinômios a uma variável  $x$ , duas variáveis  $x, y$ , etc..

Um anel é um PID ("principal ideal domain") se todo o elemento do anel admite uma factorização única por elementos primos ~~entre~~.  
 Nos polinômios e dirímos elementos  $a, b$  tem sempre mdc(1).  
 Um anel  $\neq R[x]$  é um PID se e só se  $R$  é um corpo.

Anéis de polinômios com duas ou mais variáveis nunca são PID's. Num PID da forma  $R[x]$  os elementos primos são os polinômios irreduíveis (i.e. os seus únicos fatores são 1 e o próprio).

Exemplos  $\mathbb{Z}[x]$ ,  $\mathbb{Z}[x, y], \dots$  são anéis de polinômios

$\mathbb{Z}_q[x]$  é um PID se e só se  $q$  é primo

### Polinômios ciclotônicos

As  $k$ -raízes da unidade (isto é, os elementos  $\mathbb{Z}$  que verificam  $z^k = 1$ ) são relevantes em Criptografia.

Por isso polinômios da forma

$$Q_k = x^k - 1$$

O polinômio ciclotônico de ordem  $k$ , é o polinômio de maior grau que divide  $Q_k$  mas não divide nenhum  $Q_n$  com  $n < k$ .

Toda a raiz diferente de 1 das  $i$ -ésimas potências de  $\omega_k$  é uma raiz primitiva de unidade de ordem  $k$ . Isto é, é um complexo falso que  $\omega_k^i$  é  $i = 0 \dots k-1$  gera todas as  $k$ -raízes da unidade.

O polinômio ciclotônico  $\Phi_k$  de ordem  $k$ , é o polinômio irreductível de maior grau que divide  $Q_k$  mas não divide nenhum  $Q_n$  com  $n < k$ .

As raízes de  $\Phi_k$  são as  $k$ -raízes primitivas da unidade; isto é, os complexos  $\omega$  tais que

$\mathbb{Z}^n$  para  $i = 0 \dots k-1$

gera todas as  $k$ -raízes da unidade.

Verifica-se

$$a) Q_k(x) = \prod_{d|k} \Phi_d(x)$$

b) Se  $k = p^m$  com  $p$  primo então

$$\Phi_k(x) = \Phi_p(x^{p^{m-1}}) = \sum_{i=0}^{p-1} (x^{p^{m-1}})^i$$

Ex: Para  $p = 2$ ,  $\Phi_2(x) = x + 1$

As raízes de  $Q_2(x^2 - 1)$  são  $\{-1^0, (-1)^1\}$

$(-1)$  é raiz de  $\Phi_2$   $\rightarrow$  raiz primitiva

Se  $k = 2^m$  então  $\Phi_k(x) = \Phi_2(x^{\frac{2^{m-1}}{2}}) =$

$$\Phi_k(x) = x^{\frac{2^{m-1}}{2}} + 1 = x^{\frac{k}{2}} + 1$$

Esta classe de polinômios, que têm suas fórmulas simples, é muito usada em criptografia.

c) Em particular, se  $m=1$ ,  $k=p$  com  $p$  primo

$$\Phi_p(x) = \sum_{i=0}^{p-1} x^i$$

Todas as  $p-1$  raízes de  $\Phi_p(x)$  são  $p$ -raízes primitivas da unidade.

Polinômios cíclicos em  $\mathbb{Z}_p$

Se  $p$  é um primo que não divide  $n$  então

$\phi_n$  é também ciclotômico sobre  $\mathbb{Z}_p$  se

$p$  é um gerador do grupo multiplicativo  $\mathbb{Z}_n^*$

Isto é

$$p^{\phi(n)} \equiv 1 \pmod{n}$$

$$p^i \not\equiv 1 \pmod{n} \quad \text{se } 0 < i < \phi(n) \text{ e } i \neq 0$$

Isto é

$$p^i \equiv 1 \pmod{n} \quad \text{se } i = 0 \pmod{\phi(n)}$$

### TEOREMA CHINÊS DOS RESTOS

Dada uma sequência de módulos

$$M_i = q_1, q_2, \dots, q_n$$

tais que  $\text{mdc}(q_i, q_j) = 1$  para  $i \neq j$

Dada uma sequência de restos

$$R = R_1, R_2, \dots, R_n$$

com  $0 \leq R_i < q_i$

determinar uma solução  $\underline{a} \in \mathbb{Z}_{q_1 \times q_2 \times \dots \times q_n}$  do sistema de  $n$  equações:

$$a \equiv R_1 \pmod{q_1}$$

$$a \equiv R_2 \pmod{q_2}$$

$$\dots a \equiv R_n \pmod{q_n}$$

O CRT diz que existe sempre uma solução deste problema.

que é única módulo  $q_1 \times q_2 \times \dots \times q_n$

Este teorema, inicialmente desenvolvido para o anel de inteiros, é ~~apenas~~ válida para todo o PID.

Novamente em anéis de polinômios com coeficientes num corpo. Seja  $R$  um anel.

Solução

$$\text{Seja } N = q_1 \times q_2 \times \dots \times q_n + c$$

$$N_i = N/q_i$$

1) usando o algoritmo de Euclides → temos que determina-se

$x_i, y_i$  tais que

$$x_i n_i + y_i q_i = 1$$

2) calcula-se  $r_i = x_i n_i$

Dafini resulta que  $r_i \equiv 1 \pmod{q_i}$

$$r_j \equiv 0 \pmod{q_j} \quad \forall j < i$$

( $e_i$  é um múltiplo de  $n_i$  e  $n_i \pmod{q_i} = 0$ )

3) Calcula-se

$$a = \sum_{i=1}^n r_i e_i \pmod{N}$$

A sequência  $e_1, e_2, \dots, e_n$  chama-se base CRT

Essencialmente o CRT estabelece um isomorfismo entre o anel "grande"  $R/N$  e o produto de anéis "pequenos"  $R/q_1 \times R/q_2 \times \dots \times R/q_n$ .

Exemplo  $R = \mathbb{Z}_2[x]$

$$N = x^{31} + 1$$

fatorizadas

$$\begin{aligned} N &= (x+1)(x^5+x^2+1)(x^5+x^3+1)(x^5+x^2+x^3+1) \\ &\quad (x^5+x^4+x^2+x+1)(x^5+x^4+x^3+x+1) \\ &\quad (x^5+x^4+x^3+x^2+1) \end{aligned}$$

Logo

$$q_1 = (x+1)$$

$$q_2 = (x^5+x^2+1)$$

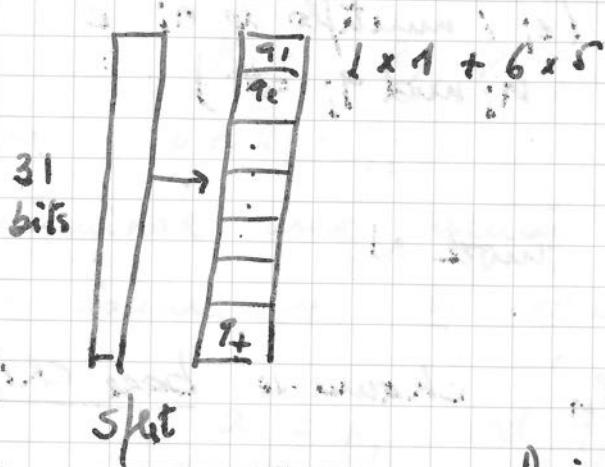
$$q_3 = (x^5+x^3+1)$$

$$q_4 = (x^5+x^3+x^2+x+1)$$

$$q_5 = (x^5+x^4+x^2+x+1)$$

$$q_6 = (x^5+x^4+x^3+x+1)$$

$$q_7 = (x^5+x^4+x^3+x^2+x+1)$$



os  $R/q_i$  são corporões

porque os  $q_i$  são polinômios irreductíveis

o  $R/N$  é apenas um anel

dado à forma de  $N$ , a soma

e multiplicação em  $R/N$  são sempre feitas de forma "

No  $(N/q_i)$  estas operações são mais complicadas

## TEOREMA CETINÊS DOS RESTOS APROXIMADO

25

Considere-se o problema dos úteiros

1) É definido um problema CRT por uma  
sequência de módulos  $q_1, q_2, \dots, q_n$  e uma  
sequência de restos:  $a_1, a_2, \dots, a_n$ .

Seja  $N = q_1 \times q_2 \times \dots \times q_n$  e seja  $\underline{a}$   
a solução do CRT exato; isto é,  $|a| < N$   
é o útiero que verifica todas as equivalências

$$a \equiv a_i \pmod{q_i} \quad \text{para todo } i \in \{1..n\}$$

2) É dado um limite  $X < N$  e uma tolerância  
 $t < n$ . Pretende-se determinar todos os  
 $\underline{x} \leq X$  que verificam simultaneamente

a)  $|x| < X$

b) Não se verifica a equivalência

$$x \equiv a_i \pmod{q_i}$$

em quanto muito  $t$  dos índices  $i \in \{1..n\}$

---

A solução  $\underline{a}$  do CRT exato encontrado em 1,  
verifica  $|a| < N$  mas não há garantias que  
satisfaca  $|a| < X$ . Por outro lado  $\underline{a}$  verifica todas  
as equivalências; no CRT aproximado dá-se a hipótese  
da solução  $\underline{x}$  falhar em afrontar das equivalências.

Obviamente se a solução exata  $\underline{a}$  verificar  $|a| < X$  ela é automaticamente uma solução do CRT aproximada. Normalmente isto não acontece e é preciso uma outra abordagem ao problema.

A solução mais imediata recorre a um teorema, descoberto por Coppersmith que procura zeros modulus de polinómios inteiros. Na sua versão mais simples o teorema diz:

### COPRERSMITH (versão 0)

Parâmetros: um inteiro  $N \geq 0$ , um factor racional positivo  $0 < \beta \leq 1$ , e um inteiro  $\underline{a}$  que verifica  $|a| < N$ .

Seja  $X$  o maior inteiro menor do que  $N^{\beta^2}$ .  
Então:

Existe um algoritmo polinomial que determina todos os inteiros  $\underline{x}$  que verificam

$$a) |x| < X$$

$$b) \text{mdc}(x - a, N) \geq N^\beta$$

Em particular, para  $\beta = 1$ , tem-se  $\underline{x} = N$

$$\text{e } \text{mdc}(x - a, N) = N ; \text{ isto é }$$

$$x - a \equiv 0 \pmod{N}$$

Para  $\beta < 1$  (por exemplo,  $\beta = \frac{1}{2}$ )  
limite  $x_L = \lfloor L^{\frac{1}{\beta}} \rfloor$

27

Tomemos agora um qualquer  $\beta < 1$  (por exemplo,  $\beta = \frac{1}{2}$ ). Então o limite é

$$x = \lfloor \sqrt[4]{N} \rfloor$$

e queremos soluções de  $\text{mdc}(x-a, N) \geq \sqrt[4]{N}$

Isto significa que tem de existir um divisor de  $N$ , digamos  $M$ , que verifica  $M \geq \sqrt[4]{N}$  e que é também divisor de  $(x-a)$ ; isto é

$$x - a \equiv 0 \pmod{M} \quad \text{ou} \quad M \text{ divide } N.$$

---

Para se aplicar estas ideias ao problema CRT aproximado, considere-se

$$N = q_1 \times \dots \times q_n$$

e  $\underline{a}$  a solução do CRT exato

$$a \equiv a_i \pmod{q_i} \quad \text{para todo } i \in \{1..n\}$$

Seja  $I$  um qualquer subconjunto dos índices  $\{1..n\}$  de tamanho pelo menos  $n-t$ . Este  $I$  é desconhecido e vai fazer parte da solução do problema aproximado. Queremos encontrar um  $\underline{x}$  que, para além de ser "pequeno" (isto é, verifica  $|x| < X$ ), verifica todas as equivalências para  $i \in I$ .

S seja  $M = \prod_{i \in I} q_i$ ; obviamente  $N$  é um divisor de  $N$ .

Resolvendo o CRT exato usando os módulos  $q_i$  com  $i \in I$ , determinamos  $\underline{x}$  tal que

$$\underline{x} \equiv a; \pmod{q_i} \quad \text{para todo } i \in I$$

Com  $\underline{x} \equiv a; \pmod{q_i} \quad \text{para todo } i \in \{1..n\}$ .

Então  $\underline{x} \equiv a \pmod{q_i} \quad \text{para todo } i \in I$

ou seja  $\underline{x} \equiv a \pmod{M}$ .

Assumimos:

- Não se conhece  $I$ , nem  $M$ , mas sabe-se

que  $M$  é um divisor de  $N$  que verifica

$$\text{máx}(x-a, N) = M$$

- Nesta saber se existe algum  $\beta \leq 1$  tal que

$$x \leq N^{\beta^2} \quad \text{e} \quad M \geq N^\beta$$

- Se for encontrado um tal  $\beta$ , ~~verifica-se~~ marcar o Teorema de CopperSmith para encontrar as várias eventuais soluções  $\underline{x}$  e para cada uma delas verificar se o número de falso não ultrapassa  $t$ .

---

Suponhamos que os  $q_i$  estão ordenados de forma crescente. Então:

$$\prod_{i=1}^{n-t} q_i \leq \prod_{i \in I} q_i = M$$

ou

$$\sum_{i=1}^{n-t} \log(q_i) \leq \log M$$

como  $\log M \geq \beta(\log N)$  e  $\log x \leq \beta^2(\log N)$   
 pode-se concluir a relação

$$\sum_{i=1}^{n-t} \log q_i \leq \beta((\log x)(\log N))$$

Esta relação permite ver, dados  $x, N$  e os  $q_i$ ,  
 qual é a tolerância  $t$  que pode ser usada.

CRT aproximado e reconstrução polinomial  
 com erros

O problema da interpolação polinomial, na sua versão  
 mais simples, é uma aplicação direta do Teorema  
 chinês dos restos em anéis de polinômios que  
 sejam PIDs.

O problema é:

Dados n pontos distintos  $x_1, x_2, \dots, x_n$   
 num corpo  $K$ , determinar e n valores  
 $y_1, y_2, \dots, y_n \in K$ , determinar o polinômio  
 de menor grau  $y(x)$  que verifica  
 $y(x_i) = y_i$  para todo  $i \in 1..n$

A solução baseia-se na observação de que, em domínios polinomiais,

$$y(x_i) \text{ coincide com } y(x) \bmod (x - x_i)$$

Por isso o problema resume-se em determinar uma solução polinomial da equação

$$y(x) \equiv y_1 \bmod (x - x_1)$$

$$y(x) \equiv y_2 \bmod (x - x_2)$$

$$\vdots$$
$$y(x) \equiv y_n \bmod (x - x_n)$$

Definidos os módulos

$$q_i(x) \equiv (x - x_i)$$

a solução deste sistema de equações é obtida pelo teorema chinês dos  $n$  restos módulos

$$N(x) \equiv \prod_{i=1}^n q_i(x)$$

Da mesma forma uma reconstrução polinomial aproximada, tem a forma

Dados  $n$  pontos  $x_1, x_2, \dots, x_n$  num corpo  $K$  e  $n$  valores  $y_1, y_2, \dots, y_n$  em  $K$ , determinar os polinômios  $y(x)$  de grau menor ou igual a  $\leq d$  (com  $d < n$ ) tais que

$$y \bmod (x - x_i) = y_i$$

em pelo menos  $n-t$  pontos.

Tal como nos cíntimos o parâmetro  $t$  determina<sup>31</sup> o numero de pontas onde existe um erro (i.e. o polinômio  $y(x)$  tem um valor diferente de  $y_i$  no ponto  $x_i$ ).

No polinômios o limite no tamanho é descrito por um gran máximo  $d$  do polinômio solução.

A formalização do problema é análoga à usada no domínio dos cíntimos

Se for  $I$  o conjunto de índices onde se verifica

$$y \text{ mod } (x - x_i) = y_i \quad i \in I$$

e se for  $y_0$  a solução do CRT exato

$$y_0 \text{ mod } (x - x_i) = y_i \quad i \in 1..n$$

e se for  $M = \prod_{i \in I} (x - x_i)$

então

$$y - y_0 = 0 \text{ mod } M$$

ou

$$\boxed{\text{mdc}(y(x) - y_0(x), N(x)) = M(x)}$$

Este problema tem uma solução usando o teorema de Coppersmith na sua versão polinomial. Especificamente o algoritmo de Coppersmith, produz para todo  $\beta \in (0, 1]$  um polinômio  $y(x)$  de

$$\boxed{\begin{array}{l} \text{grau } \deg_x Y < \beta^2 n \text{ tal que} \\ \deg_x \text{ mdc}(Y(x) - Y_0(x), N(x)) \geq \beta n \end{array}}$$

No exemplo típico entao  $\deg(M(x)) = n-t \geq \beta n$

$$\text{e } d = \beta^2 n.$$

On seja

$$\frac{d}{n} = \beta^2 \quad \text{é desejado por \underline{code rate}}$$

$$\frac{n-t}{n} \geq \beta \quad \text{com } \frac{t}{n} \text{ o \underline{error rate}}$$

A reescrivendo os 3 parâmetros é entao

$$\boxed{\frac{t}{n} \leq 1 - \sqrt{\frac{d}{n}}}$$

O algoritmo de Guruswami-Sudan é uma boa aproximação do algoritmo de Coppersmith binomial.