D34: Méthodes de calcul efficaces et sécurisées

Nicolas Méloni

Master 2: 1er semestre

(2014/2015)

Exponentiation modulaire

Définition

- $x^k = x * \cdots * x$ où x est l'élément d'un groupe
- Opération principale des primitives cryptographiques comme RSA ou El Gamal
- Chiffrement/déchiffrement RSA: $x^k \mod n$

Méthode générale

- On considère la décomposition en base 2 de l'exposnant $k = (k_{t-1} \ldots k_1 k_0)_2$ de sorte que

$$x^k = x^{k_{t-1}2^{t-1} + \ldots k_1 2 + k_0} = x^{k_{t-1}2^{t-1}} \ldots x^{k_1 2} x^{k_0}$$

- Asymptotiquement: $\log_2(k)$ multiplications

Square-and-Multiply

Algorithm 1 Right to left Square-and-multiply

Require: $x \in G$, $k = (k_{t-1} \ldots k_1 k_0)_2$

Ensure: x^k

```
y ← 1
for i = 0 ... t − 1 do
    if k_i = 1 then
        y ← yx
    end if
    x ← x^2
end for
return y
```

Square-and-Multiply

Algorithm 2 Left to right Square-and-multiply

Require: $x \in G$, $k = (k_{t-1} \dots k_1 k_0)_2$

Ensure: x^k

```
y ← 1
for i = t − 1 … 0 do
    y ← y^2
    if k_i = 1 then
        y ← yx
    end if
end for
return y
```

Analyse

- L'algorithme effectue t élévations au carré ($\mathtt{S}$) et $\mathtt{wt}(k)$ multiplications ($\mathtt{M}$).
- Compléxité moyenne: $\lfloor \log_2(k) \rfloor \mathtt{S} + \frac{\lfloor \log_2(k) \rfloor}{2} \mathtt{M}$.
- On ne peut pas diminuer le nombre de carrés, par contre on peut essayer d'optimiser le nombre de multiplications.

Définition

- On appelle chaîne d'addition calculant k de longueur l, toute suite (a_i) finie telle que:

$$a_1 = 1, a_l = k, \forall i \text{ tel que } 2 \leq i \leq l, \exists (j_1, j_2) : a_i = a_{j_1} + a_{j_2}.$$

Exemple

- Chaînes calculant 33:

$$\begin{aligned} C_1 &= (1, 2, 3, 4, \ldots, 32, 33) \\ C_2 &= (1, 2, 3, 4, 7, 10, 13, 20, 33) \\ C_3 &= (1, 2, 4, 8, 16, 32, 33) \end{aligned}$$

Chaînes d'additions

- Une chaînes d'addition calculant k fournie naturellement un algorithme d'exponentiation modulaire calculant x^k
- Trouver la chaîne d'addition la plus courte calculant un une suite d'entier $(k_1, \ldots, k_t)$ est un problème NP complet
- Dans le cas $t = 1$ il existe des algorithmes très efficaces pour obtenir des chaînes relativement courtes
- Pour presque tout n, il existe une chaîne d'addition calculant n de longueur $\log(n) + (1 + o(1))\frac{\log(n)}{\log\log(n)}$.

Algorithme de Brauer

Idée générale

- Précalculer un certain nombre de puissance de x au début et les réutiliser en cours d'exponentiation
- Considérer pour cela l'écriture de k en base 2^w pour un certain $w > 1$:

$$k = \sum_{i=0}^{t-1} k_i (2^w)^i, k_i \in \{0, \ldots, 2^w - 1\}.$$

ALgorihtme de Brauer

Algorithm 3 Exponentiation de Brauer

Require: $x \in G$, $k = (k'_l \dots k'_1 k'_0)_{2^w}$
Ensure: x^k

$y \leftarrow 1, x_0 \leftarrow 1$
for $i = 1 \dots w - 1$ **do**
    $x_i \leftarrow x_i * x$
end for
$y \leftarrow x_{k_l}$
for $i = l - 1 \dots 0$ **do**
    $y \leftarrow y^{2^w}$
    $y \leftarrow y * x_{k_i}$
end for
return y

Analyse

- On a $l = \left\lfloor \frac{\log_2(k)}{w} \right\rfloor$,
- L'algorithme effectue $2^w - 1$ multiplications dans la première boucle puis wl élévations au carré (`S`) et $l+1$ multiplications dans la deuxième (`M`).
- Complexité moyenne:

$$w \left\lfloor \frac{\log_2(k)}{w} \right\rfloor \mathtt{S} + (2^w + \left\lfloor \frac{\log_2(k)}{w} \right\rfloor)\mathtt{M}.$$

- Optimal pour w de l'ordre de $\log \log(k)$

Optimisation de l'algorithme de Brauer

Éliminer les multiplications par 1!

- C'est le cas lorsque $k_i = 0$

Éliminer les digits pairs

- L'idée est de ne pas précalculer les entiers $2, 4, \ldots, 2k - 2$ et de remplacer les séquences *doublement puis ajouter* r par *ajouter* $r/2$ *puis doublement*.

Faire varier les exposants (sliding windows)

- L'algorithme impose que chaque puissance de 2 a pour exposant jw. On economise plusieurs multiplications en autorisant plus de fléxibilité sur ces exposants.

Non-Adjacent-Form

- On considère un système de représentation des nombres pour lequel les chiffres peuvent prendre des valeurs négatives.
- Avec les chiffres $\{-1, 0, 1\}$ il existe 3^n representations différentes pour seulement 2^n entiers.
- On cherche les représentation les plus creuses (i.e. avec le plus de 0)

Algorithm 4 Conversion NAF

Require: $k \in \mathbb{N}$
Ensure: $\texttt{NAF}(k)$

$i \leftarrow 0$
while $k \leq 1$ **do**
  if $k \mod 2 = 1$ **then**
    $k_i \leftarrow 2 - (k \mod 4)$
    $k \leftarrow k - k_i$
  else
    $k_i \leftarrow 0$
  end if
  $k \leftarrow k/2, i \leftarrow i + 1$
end while
return $(k_{i-1} \ldots k_1 k_0)$

Non-Adjacent-Form

Algorithm 5 Exponentiation NAF

Require: $k \in \mathbb{N}, x \in G$
Ensure: x^k

$(k_{l-1} \dots k_0) \leftarrow \mathbb{NAF}(k)$
$x_1 \leftarrow x, x_{-1} \leftarrow x^{-1}, y \leftarrow 1$
for $i = l - 1 \dots 0$ **do**
    $y \leftarrow y^2$
    if $k_i \neq 0$ **then**
        $y \leftarrow y * x_{k_i}$
    end if
end for
return y

Propriété

- Soit k un entier et $(k_{l-1} \dots k_0)$ sa représentation `NAF`. Alors pour tout $0 \leq i \leq l-2$, $k_{i+1}k_i = 0$.

Analyse

- Densité de chiffres non-nuls: 1/3
- Compléxité moyenne: $(\lfloor \log_2(k) \rfloor + 1)\mathtt{S} + \frac{\lfloor \log_2(k) \rfloor + 1}{3}\mathtt{M}$.
- De manière similaire on peut diminuer le nombres de multiplications en considèrant un ensemble de chiffres plus large: $\{-2^{w-1}+1, \dots -1, 0, 1, \dots, 2^{w-1}-1\}$.

Algorithm 6 Conversion w-NAF

Require: $k, w \in \mathbb{N}$
Ensure: $\mathtt{NAF}_w(k)$

$i \leftarrow 0$
while $k \leq 1$ **do**
  if $k \mod 2 = 1$ **then**
    $k_i \leftarrow k \mod 2^w)$
    $k \leftarrow k - k_i$
  else
    $k_i \leftarrow 0$
  end if
  $k \leftarrow k/2, i \leftarrow i + 1$
end while
return $(k_{i-1} \ldots k_1 k_0)$

- Les algorithmes précédents sont des généralisations de l'algorithmes `square-and-multiply` de gauche à droite.
- L'algorithme de Yao en est le pendant droite à gauche.
- Soit $k = k_{l-1}2^{l-1} + \cdots + k_1 2 + k_0$ avec $k_i \in \{0, 1, \ldots, 2^w - 1\}$, l'idée principale consiste à réécrire k sous la forme:

$$1 \times \underbrace{\sum_{k_i=1} 2^i}_{d(1)} + 2 \times \underbrace{\sum_{k_i=2} 2^i}_{d(2)} + \cdots + (2^w - 1) \times \underbrace{\sum_{k_i=2^w-1} 2^i}_{d(2^w-1)} .$$

Algorithme de Yao

Algorithm 7 Exponentiation de Yao

Require: $k = (k_{l-1} \dots k_0)_{2^w} \in \mathbb{N}, x \in G$
Ensure: x^k
for $i = 1 \dots 2^w - 1$ **do** $x_{d(i)} \leftarrow 1$
end for
for $i = 0 \dots l - 1$ **do**
    if $k_i \neq 0$ **then**
        $x_{d(k_i)} \leftarrow x_{d(k_i)} x$
    end if
    $x \leftarrow x^2$
end for
$y \leftarrow 1, a \leftarrow 1$
for $i = 2^w - 1 \dots 0$ **do**
    $a \leftarrow a x_{d(i)}$
    $y \leftarrow y a$
end for
return y

Exemple

- Prenons $k = 314159 = 100\,0300\,1003\,0000\,5007$, $l = 19$ et $2^w - 1 = 7$.
  $k = 1 \times (2^{18} + 2^{11}) + 3 \times (2^{14} + 2^8) + 5 \times 2^3 + 7 \times 2^0$

$$
\begin{aligned}
d(1) &= 100\,0000\,1000\,0000\,0000 \\
d(3) &= 000\,0100\,0001\,0000\,0000 \\
d(5) &= 000\,0000\,0000\,0000\,1000 \\
d(7) &= 000\,0000\,0000\,0000\,0001 \\
k &= 100\,0300\,1003\,0000\,5007 \\
&= 7d(7) + 5d(5) + 3d(3) + d(1)
\end{aligned}
$$

Algorithme de Yao

Analyse

- Nombres d'opérations identiques à l'agorithme de Brauer
- Compatible avec toute les améliorations classiques de l'algorithme de Brauer.
- En général légèrement plus lent à cause des accés mémoires.

Exponentiation d'un élément fixe

- Dans de nombreux protocoles, l'élément x est connu à l'avance (ex: Diffie-Hellman)
- Il est alors possible de précalculer certaines valeurs de x afin de diminuer le cout final de l'exponentiation
- Par exemple, en précalculant les valeurs $x^2, x^{2^2}, \ldots, x^{2^l}$, l'algorithme `square-and-multiply` de droite à gauche effectue alors en moyenne $\frac{\lfloor \log_2(k) \rfloor}{2}\mathtt{M}$.

Algorithme de Yao pour élément fixe

- Les algorithmes de droite à gauche s'adaptent naturellement au cas de l'exponentiation d'un élément fixe.

Algorithm 8 Exponentiation de Yao avec élément fixe

Require: $k = (k_{l-1} \dots k_0)_{2^w} \in \mathbb{N}, x \in G$
Ensure: x^k
1. (précalculs) $x_i = x^{2^{wi}}$
2. **for** $i = 0 \dots l - 1$ **do**
3.   **if** $k_i \neq 0$ **then**
4. $x_{d(k_i)} \leftarrow x_{d(k_i)} x_i$
5.   **end if**
6. **end for**
7. $y \leftarrow 1, a \leftarrow 1$
8. **for** $i = 2^w - 1 \dots 0$ **do**
9. $a \leftarrow a x_{d(i)}$
10. $y \leftarrow ya$
11. **end for**
12. **return** y

Analyse

- Stockage: $\lceil \log_2(k)/w \rceil$ éléments.
- Nombre d'opérations: $2^w + \lceil \log_2(k)/w \rceil$ multiplications.

- On décompose k sous la forme $K_{l-1}||\dots||K_0$ où les K_i sont chaînes de t bits.
- On écrit les K_i en colonne:

$$\begin{bmatrix} K_0 \\ K_1 \\ \vdots \\ K_{l-2} \\ K_{l-1} \end{bmatrix} = \begin{bmatrix} k_{t-1} & \dots & k_0 \\ \vdots & & \vdots \\ k_{dt-1} & \dots & k_{(d-1)t} \\ \vdots & & \vdots \\ k_{lt-1} & \dots & k_{(l-1)t} \end{bmatrix}$$

- On précalcule les $x_{[b_{l-1},\dots,b_0]} = x^B$ où $B = b_{l-1}2^{(l-1)t} + \dots + b_d 2^{(d)t} + \dots + b_1 2^t + b_0$, pour toute les chaînes de bits $b_{l-1}\dots b_0$.

Méthode combinée

Algorithm 9 Méthode combinée

Require: $k = (k_{lt-1} \dots k_0) \in \mathbb{N}, x \in G$

Ensure: x^k

1. (précalculs) $x_{[b_{l-1},\dots,b_0]} = x^B$ où $B = b_{l-1}2^{(l-1)t} + \dots + b_1 2^t + b_0$
2. $y \leftarrow 1$
3. **for** $i = 0 \dots t-1$ **do**
4. $\quad y \leftarrow y^2$
5. $\quad y \leftarrow y x_{[k_{lt-1-i},\dots,k_{t-1-i}]}$
6. **end for**
7. **return** y

Analyse

- Stockage: 2^l puissances de x ($l = \lceil \log_2(k)/t \rceil$)
- Nombre d'opérations: $t\mathtt{M} + t\mathtt{S}$
- En moyenne: $\left(\frac{2^l-1}{2^l}\right) t\mathtt{M} + t\mathtt{S}$

Définition

- Soit k un entier positif. On appelle représentation en base double de k toute écriture de k sous la forme

$$\sum_{i=0}^{n} 2^{b_i} 3^{t_i}.$$

Exemple

$$\begin{aligned}
127 &= 2^2 3^3 + 2^1 3^2 + 2^0 3^0 \\
&= 2^5 3^1 + 2^2 3^0 + 2^0 3^3 \\
&= 2^4 3^0 + 2^2 3^3 + 2^1 3^0 + 2^0 3^0 \\
&= 2^2 3^3 + 2^2 3^1 + 2^2 3^0 + 2^1 3^0 + 2^0 3^0
\end{aligned}$$

Double-base Number System (DBNS)

Propriété

- Pour tout k suffisament il existe une représentation en base double dont le nombre de termes non nuls est de l'ordre de $O\left(\frac{\log(n)}{\log\log(n)}\right)$.

Exponentiation avec base double

- Précaluler tous les $x_{ij} = x^{2^i 3^j}$
- Obtenir une représentation DBNS de k
- Effectuer le produit des x_{ij}

Multi-exponentiation

- Certains protocoles nécessitent d'effectuer le calcul $x^k y^l$
- Une méthode évidente consiste à calculer x^k et y^l séparément puis à effectuer un produit final.
- Pour accélérer les calculs on va chercher un algorithme permettant d'obtenir directement $x^k y^l$

Shamir's trick

- On considère simultanément les bits de k et l et on effectue une version légèrement adaptée de l'algorithme `square-and-multiply`

Multi-exponentiation

Algorithm 10 Multi `square-and-multiply`

Require: $k = (k_{t-1} \dots k_0), l = (l_{t-1} \dots l_0) \in \mathbb{N}, x, y \in G$
Ensure: $x^k y^k$

```
z ← 1
for i = t − 1 ... 0 do
    z ← z²
    if k_i = l_i = 1 then z ← z(xy)
    else if k_i = 1 then z ← zx
    else if l_i = 1 then z ← zy
    end if
end for
return z
```

Analyse

- Nombre d'opérations: $t\mathtt{M} + t\mathtt{S}$
- En moyenne: $\frac{3}{4}t\mathtt{M} + t\mathtt{S}$

Multi-exponentiation

- Comme pour l'exponentiation simple, on peut considèrer la représentation en base 2^w pour diminuer le nombre de mutliplications.
- On considère les écritures $k = K_{d-1}||\ldots||K_0$, $l = L_{d-1}||\ldots||L_0$ où les K_i, L_i sont des chaînes de w bits.
- L'exponentiation commence par le précalcul de $x^i y^j$ pour tout les i, j dans $[0, 2^w - 1]$

Multi-exponentiation

Algorithm 11 Multi exponentiation

Require: une taille de fenêtre w, $k = (K_{d-1}||\dots||K_0)$, $l = (L_{d-1}||\dots||L_0) \in \mathbb{N}, x, y \in G$

Ensure: $x^k y^k$

1. Pour tout les $i, j \in [0, 2^w - 1]$, $z_{ij} \leftarrow x^i y^j$
2. $z \leftarrow 1$
3. **for** $i = d - 1 \dots 0$ **do**
4. $\quad z \leftarrow z^2$
5. $\quad$ **if** $K_i \neq 0$ ou $L_i \neq 0$ **then**
6. $\quad\quad z \leftarrow z(z_{K_i L_i})$
7. $\quad$ **end if**
8. **end for**
9. **return** z

Analyse

- Phase de précalcul: $3 \times 2^{2(w-1)}\mathtt{M} + 2^{2(w-1)}\mathtt{S}$
- Nombre d'opérations moyen: $\frac{2^{2w}-1}{2^{2w}} d\mathtt{M} + d\mathtt{S}$