Enhanced Digital Signature using RNS Digit Exponent Representation

Thomas Plantard[(1)], Jean-Marc ROBERT[(2)]

1: CCISR, SCIT, University of Wollongong, Australia
2: Team DALI/LIRMM, University of Perpignan, France

15 July 2016

Work funded by ANR PAVOIS 12 BS02 002 02 project

Outline

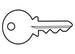
Outline

Outline

DSA Signature

Bob signs a message to Alice :-)

$$\mathcal{S}, M \rightarrowtail$$



$x \rightarrow$ Bob's private key

$y \rightarrow$ Bob's public key

Group $\mathcal{G}$	$((\mathbb{Z}/p\mathbb{Z})^*, \times, 1)$
Bob hashes the message M, using an approved hash function, $\Rightarrow z = \mathcal{H}(M)$;	
Bob's public parameters	a prime $p \in \mathbb{N}$ a generator g of $\mathcal{G}$ of order q
Bob's private key	$x \in [1, q-1]$
Bob's public key	$y = g^x \bmod p$

DSA Signature (2)

Bob signs a message to Alice :-)

$$\overset{\mathcal{S},\, M}{\rightarrowtail}$$

Bob chooses a random parameter
$k \in [1, q-1]$

Signature (Bob)
$\mathcal{S} = (r, s)$

$$\text{with} \left\{ \begin{array}{lcl} r & = & (g^k \bmod p) \bmod q \\ s & = & (k^{-1}(z + xr)) \bmod q \end{array} \right.$$

$\mathcal{V}erif(\mathcal{S})$:
Alice receives (r', s') and M'.

$$\left\{ \begin{array}{c} w = (s')^{-1} \bmod q \\ z = \mathcal{H}(M') \\ u_1 = (zw) \bmod q \\ u_2 = ((r')w) \bmod q \\ u = ((g^{u_1} y^{u_2} \bmod p) \bmod q) \end{array} \right.$$

$\rightarrow$ Alice checks wether $u = r'$.

DSA Signature (2)

Bob signs a message to Alice :-)

$$\mathcal{S}, M \atop \rightarrowtail$$

Bob chooses a random parameter
$k \in [1, q-1]$

Signature (Bob)
$\mathcal{S} = (r, s)$

with $\begin{cases} r & = & (g^k \bmod p) \bmod q \\ s & = & (k^{-1}(z + xr)) \bmod q \end{cases}$

$\mathcal{V}erif(\mathcal{S})$:
Alice receives (r', s') and M'.

$$\begin{cases} w = (s')^{-1} \bmod q \\ z = \mathcal{H}(M') \\ u_1 = (zw) \bmod q \\ u_2 = ((r')w) \bmod q \\ u = ((g^{u_1} y^{u_2} \bmod p) \bmod q) \end{cases}$$

$\rightarrow$ Alice checks wether $u = r'$.

$\rightarrow$The main operation is the modular exponentiation $g^k \bmod p$.

Outline

Square-and-Multiply

Left-to-Right Square-and-Multiply Modular Exponentiation

Require: $k = (k_{t-1}, \ldots, k_0)$, the DSA modulus p, g a generator of $\mathbb{Z}/p\mathbb{Z}$ of order q.
Ensure: $X = g^k \bmod p$

$X \leftarrow 1$
for i from $t-1$ downto 0 **do**
  $X \leftarrow X^2 \bmod p$
  if $k_i = 1$ **then**
    $X \leftarrow X \cdot g \bmod p$
  end if
end for
return (X)

Square-and-Multiply

Left-to-Right Square-and-Multiply Modular Exponentiation

Require: $k = (k_{t-1}, \ldots, k_0)$, the DSA modulus p, g a generator of $\mathbb{Z}/p\mathbb{Z}$ of order q.
Ensure: $X = g^k \bmod p$

$X \leftarrow 1$
for i from $t-1$ downto 0 **do**
  $X \leftarrow X^2 \bmod p$
  if $k_i = 1$ **then**
    $X \leftarrow X \cdot g \bmod p$
  end if
end for
return (X)

No storage, $t-1$ squarings, $\approx \frac{t}{2}$ multiplications.

Radix-R

Radix-R Exponentiation Method (Gordon, 1998)

Require: $k = (k_{\ell-1}, \dots, k_0)_R$, the DSA modulus p, g a generator of $\mathbb{Z}/p\mathbb{Z}$ of order q.

Ensure: $X = g^k \bmod p$

Precomputation. Store $G_{i,j} \leftarrow g^{i \cdot R^j}$, with $i \in [1, ..., R-1]$ and $0 \leq j < \ell$.

$X \leftarrow 1$

for i from $\ell - 1$ downto 0 **do**

$X \leftarrow X \cdot G_{k_i,i} \bmod p$

end for

return (X)

Radix-R

Radix-R Exponentiation Method (Gordon, 1998)

Require: $k = (k_{\ell-1}, \ldots, k_0)_R$, the DSA modulus p, g a generator of $\mathbb{Z}/p\mathbb{Z}$ of order q.
Ensure: $X = g^k \bmod p$
Precomputation. Store $G_{i,j} \leftarrow g^{i \cdot R^j}$, with $i \in [1, \ldots, R-1]$ and $0 \leq j < \ell$.
$X \leftarrow 1$
for i from $\ell - 1$ downto 0 **do**
  $X \leftarrow X \cdot G_{k_i,i} \bmod p$
end for
return (X)

With $w \leftarrow \log_2(R)$ $\rightarrow$ Storage of $\lceil t/w \rceil \cdot (R-1)$ values $\in \mathbb{F}_p$,
no squarings, $\ell = \lceil t/w \rceil$ multiplications.

Fixed-base Comb Method

Fixed-base Comb Method (Lim & Lee, Crypto '94)

Require: $k = (k_{t-1}, \ldots, k_1, k_0)_2$, the DSA modulus p, g a generator of $\mathbb{Z}/p\mathbb{Z}$ of order q, window width w, $d = \lceil t/w \rceil$.

Ensure: $X = g^k \bmod p$

By padding k on the left by 0's if necessary, write $k = K^{w-1} \| \ldots \| K^1 \| K^0$, where each K^j is a bit string of length d. Let K_i^j denote the i^{th} bit of K^j.

$X \leftarrow 1$

for i from $d-1$ downto 0 **do**

  $X \leftarrow X^2 \bmod p$

  $X \leftarrow X \cdot g^{[K_i^{w-1}, \ldots, K_i^1, K_i^0]} \bmod p$

end for

return (X)

Fixed-base Comb Method

Fixed-base Comb Method (Lim & Lee, Crypto '94)

Require: $k = (k_{t-1}, \ldots, k_1, k_0)_2$, the DSA modulus p, g a generator of $\mathbb{Z}/p\mathbb{Z}$ of order q, window width w, $d = \lceil t/w \rceil$.
Ensure: $X = g^k \bmod p$
By padding k on the left by 0's if necessary, write $k = K^{w-1} \| \ldots \| K^1 \| K^0$, where each K^j is a bit string of length d. Let K^j_i denote the i^{th} bit of K^j.
$X \leftarrow 1$
for i from $d-1$ downto 0 **do**
  $X \leftarrow X^2 \bmod p$
  $X \leftarrow X \cdot g^{[K_i^{w-1}, \ldots, K_i^1, K_i^0]} \bmod p$
end for
return (X)

With $d \leftarrow \lceil t/w \rceil$ $\rightarrow$ Storage of $2^w - 1$ values $\in \mathbb{F}_p$,
$d-1$ squarings, d multiplications.

Synthesis

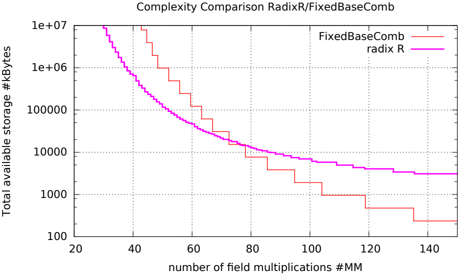
Complexities and storage amounts of state of the art methods, average case.

	# MM	# MS	storage (# values $\in \mathbb{F}_p$)
Square-and-multiply	$t/2$	$t-1$	-
Radix-R method	$\lceil t/w \rceil$	-	$\lceil t/w \rceil \cdot (R-1)$
Fixed-base Comb	$d = \lceil t/w \rceil$	$d-1$	$2^w - 1$

Synthesis

Complexities and storage amounts of state of the art methods, average case.

	# MM	# MS	storage (# values $\in \mathbb{F}_p$)
Square-and-multiply	$t/2$	$t-1$	-
Radix-R method	$\lceil t/w \rceil$	-	$\lceil t/w \rceil \cdot (R-1)$
Fixed-base Comb	$d = \lceil t/w \rceil$	$d-1$	$2^w - 1$



key size $t = 512$ bits (MS $= 0.86 \times$ MM).

Outline

Outline

Contributions

Starting from the Radix-R method:

- RNS digit recoding for exponent;

Contributions

Starting from the Radix-R method:

- RNS digit recoding for exponent;
- Enhanced algorithm for modular exponentiation;

Contributions

Starting from the Radix-R method:

- RNS digit recoding for exponent;
- Enhanced algorithm for modular exponentiation;
- Complexity and storage requirements evaluation;

Contributions

Starting from the Radix-R method:

- RNS digit recoding for exponent;
- Enhanced algorithm for modular exponentiation;
- Complexity and storage requirements evaluation;
- Software implementations, showing performance improvements.

Outline

Recoding Algorithm

The Radix-$R = m_0 \cdot m_1$ representation is as follows ($gcd(m_0, m_1) = 1$):

$$k = \sum_{i=0}^{\ell-1} k_i R^i, \text{ with } \ell = \lceil t / \log_2(R) \rceil,$$

and we represent the digits k_i using RNS with base $\mathcal{B} = \{m_0, m_1\}$:

$$\begin{cases} k_i^{(0)} = k_i \bmod m_0 = |k_i|_{m_0}, \\ k_i^{(1)} = k_i \bmod m_1 = |k_i|_{m_1}. \end{cases}$$

Recoding Algorithm

The Radix-$R = m_0 \cdot m_1$ representation is as follows ($gcd(m_0, m_1) = 1$):

$$k = \sum_{i=0}^{\ell-1} k_i R^i, \text{ with } \ell = \lceil t / \log_2(R) \rceil,$$

and we represent the digits k_i using RNS with base $\mathcal{B} = \{m_0, m_1\}$:

$$\begin{cases} k_i^{(0)} = k_i \bmod m_0 = |k_i|_{m_0}, \\ k_i^{(1)} = k_i \bmod m_1 = |k_i|_{m_1}. \end{cases}$$

Chinese Remainder Theorem

Using the CRT, one can retrieve k_i:

$$k_i = \left| k_i^{(0)} \cdot m_1 \cdot |m_1^{-1}|_{m_0} + k_i^{(1)} \cdot m_0 \cdot |m_0^{-1}|_{m_1} \right|_R .$$

Recoding Algorithm

In the sequel, let's denote (when $k_i^{(1)} \neq 0$)

$$\left.\begin{array}{l} m'_0 = m_1 \cdot |m_1^{-1}|_{m_0}, \\ m'_1 = m_0 \cdot |m_0^{-1}|_{m_1}, \\ k'_i = |k_i^{(0)} \cdot (k_i^{(1)})^{-1}|_{m_0}. \end{array}\right\}$$

Recoding Algorithm

In the sequel, let's denote (when $k_i^{(1)} \neq 0$)

$$\left.\begin{array}{l} m'_0 = m_1 \cdot |m_1^{-1}|_{m_0}, \\ m'_1 = m_0 \cdot |m_0^{-1}|_{m_1}, \\ k'_i = |k_i^{(0)} \cdot (k_i^{(1)})^{-1}|_{m_0}. \end{array}\right\} \text{Recoding: } \rightarrow \kappa_i \leftarrow (k'_i, k_i^{(1)})$$

We then rewrite the CRT, with the modular reduction $\bmod R$, as follows:

"New" Chinese Remainder Theorem

$$k_i = k_i^{(1)} |k'_i \cdot m'_0 + m'_1|_R - \lfloor k_i^{(1)} \cdot |k'_i \cdot m'_0 + m'_1|_R / R \rfloor \cdot R.$$

Recoding Algorithm

In the sequel, let's denote (when $k_i^{(1)} \neq 0$)

$$\left.\begin{array}{l} m'_0 = m_1 \cdot |m_1^{-1}|_{m_0}, \\ m'_1 = m_0 \cdot |m_0^{-1}|_{m_1}, \\ k'_i = |k_i^{(0)} \cdot (k_i^{(1)})^{-1}|_{m_0}. \end{array}\right\} \text{Recoding: } \rightarrow \kappa_i \leftarrow (k'_i, k_i^{(1)})$$

We then rewrite the CRT, with the modular reduction $\bmod R$, as follows:

"New" Chinese Remainder Theorem

$$k_i = k_i^{(1)} |k'_i \cdot m'_0 + m'_1|_R - \overbrace{\lfloor k_i^{(1)} \cdot |k'_i \cdot m'_0 + m'_1|_R / R \rfloor}^{c} \cdot R.$$

Recoding Algorithm

In the sequel, let's denote (when $k_i^{(1)} \neq 0$)

$$\left.\begin{aligned} m_0' &= m_1 \cdot |m_1^{-1}|_{m_0}, \\ m_1' &= m_0 \cdot |m_0^{-1}|_{m_1}, \\ k_i' &= |k_i^{(0)} \cdot (k_i^{(1)})^{-1}|_{m_0}. \end{aligned}\right\} \text{Recoding: } \rightarrow \kappa_i \leftarrow (k_i', k_i^{(1)})$$

We then rewrite the CRT, with the modular reduction $\bmod R$, as follows:

"New" Chinese Remainder Theorem

$$k_i = k_i^{(1)} |k_i' \cdot m_0' + m_1'|_R - \overbrace{\lfloor k_i^{(1)} \cdot |k_i' \cdot m_0' + m_1'|_R / R \rfloor}^{C} \cdot R.$$

C is a carry ($0 \leq C < m_1$):

$$\begin{cases} \text{if } k_{i+1} \geq C & \text{then} \quad k_{i+1} \leftarrow k_{i+1} - C, C \leftarrow 0, \\ & \text{else} \quad k_{i+1} \leftarrow k_{i+1} + R - C, C \leftarrow 1, \end{cases}$$

and one gets $k_{i+1} \geq 0$.

Recoding Algorithm

When $k_i^{(1)} = 0$:

$k_i = (|k_i^{(0)} + 1|_{m_0} \cdot m'_0 - m'_0)$,
thus keeping $\kappa_i \leftarrow (|k_i^{(0)} + 1|_{m_0}, 0)$ as a representation of k_i in this case.

- C is not modified here (it is either 0 or 1 and has been previously settled).
- it might be necessary to process a last carry C, with a final correction.

Recoding Algorithm

When $k_i^{(1)} = 0$:

$k_i = (|k_i^{(0)} + 1|_{m_0} \cdot m'_0 - m'_0)$,
thus keeping $\kappa_i \leftarrow (|k_i^{(0)} + 1|_{m_0}, 0)$ as a representation of k_i in this case.

- C is not modified here (it is either 0 or 1 and has been previously settled).
- it might be necessary to process a last carry C, with a final correction.

$\rightarrow$ The sequence of the $\kappa_i \leftarrow (k'_i, k_i^{(1)})$ is the $m_0 m_1$ recoding of k.

Recoding Algorithm

$m_0 m_1$ Recoding

Require: $\{m_0, m_1\}$ RNS base with $R = m_0 \cdot m_1$, $k = \sum_{i=0}^{\ell-1} k_i R^i$.
Ensure: $\{\kappa_i, 0 \le i < \ell, (C)\}$, $m_0 m_1$ recoding of scalar k.

```
C ← 0
for i from 0 to ℓ − 1 do
    k_i ← k_i − C, C ← 0
    if k_i < 0 then
        k_i ← k_i + R, C ← 1
    end if
    k_i^(0) = |k_i|_{m0}, k_i^(1) = |k_i|_{m1}.
    if k_i^(1) = 0 then
        κ_i ← (|k_i^(0) + 1|_{m0}, 0)
    else
        k'_i ← |k_i^(0) · (k_i^(1))^(−1)|_{m0}
        C ← C + ⌊k_i^(1) · |k'_i · m'_0 + m'_1|_R / R⌋
        κ_i ← (k'_i, k_i^(1))
    end if
end for
return {κ_i, 0 ≤ i < ℓ, (−C)}
```

Recoding Algorithm Example

Example: $\mathcal{B} = \{11, 8\}$ (i.e. $m_0 = 11, m_1 = 8$), $R = m_0 \cdot m_1 = 88$, $\ell = \lceil 20/\log_2(88) \rceil = 4$, and therefore

$$m'_0 = 8 \cdot |8^{-1}|_{11} = 56,$$
$$m'_1 = 11 \cdot |11^{-1}|_8 = 33.$$

Let us take $k = 936192_{10}$ (exponent size t of 20 bits, $0 < k < 2^{20}$)

$$k = 48 + 78 \cdot 88 + 32 \cdot 88^2 + 1 \cdot 88^3.$$

Recoding Algorithm Example

Let us take $k = 936192_{10}$ (exponent size t of 20 bits, $0 < k < 2^{20}$)

$$k = 48 + 78 \cdot 88 + 32 \cdot 88^2 + 1 \cdot 88^3.$$

`for` loop, steps 2 to 15:

- In the first iteration ($i = 0$), one has $\kappa_0 = k_0 = 48$.
  - One has $C \leftarrow 0$ and one skips the `if`-test steps 4 to 6 since $k_0 \geq 0$.
  - Step 7, one computes the RNS representation in base $\mathcal{B}$ of $\kappa_0 = 48$:

$$k_0^{(0)} = |k_0|_{11} = 4, \; k_0^{(1)} = |k_0|_8 = 0.$$

  - Steps 6 and 7, since $k_0^{(1)} = 0$, one sets

$$\kappa_0 \leftarrow (|k_0^{(0)} + 1|_{11}, 0) = (5, 0)$$

Values returned by the algorithm:

$$\kappa = ((5, 0), \qquad\qquad\qquad), \text{ and}$$

Recoding Algorithm Example

Let us take $k = 936192_{10}$ (exponent size t of 20 bits, $0 < k < 2^{20}$)

$$k = 48 + 78 \cdot 88 + 32 \cdot 88^2 + 1 \cdot 88^3.$$

`for` loop, steps 2 to 15:

- In the second iteration ($i = 1$), one has $\kappa_1 = k_1 = 78$.
  - One has $C \leftarrow 0$ and one skips the `if`-test steps 4 to 6 since $k_1 \geq 0$.
  - Step 7, one computes the RNS representation in base $\mathcal{B}$ of $\kappa_1 = 78$:

$$k_1^{(0)} = |k_1|_{11} = 1,\ k_1^{(1)} = |k_1|_8 = 6.$$

  - Steps 11 and 12, since $k_1^{(1)} \neq 0$, one has

$$\begin{array}{lcl} |(k_1^{(1)})^{-1}|_{11} & \leftarrow & 2 \\ k_1' = |k_1^{(0)} \cdot (k_1^{(1)})^{-1}|_{11} & \leftarrow & 2 \\ C \leftarrow \lfloor (k_1^{(1)} \cdot |k_1' \cdot 56 + 33|_{88})/88 \rfloor & \leftarrow & 3 \end{array}$$

  and finally

$$\kappa_1 \leftarrow (2, 6)$$

Values returned by the algorithm:

$$\kappa = ((5, 0), (2, 6), \qquad\qquad), \text{ and}$$

Recoding Algorithm Example

Let us take $k = 936192_{10}$ (exponent size t of 20 bits, $0 < k < 2^{20}$)

$$k = 48 + 78 \cdot 88 + 32 \cdot 88^2 + 1 \cdot 88^3.$$

`for` loop, steps 2 to 15:

- In the third iteration ($i = 2$), one has now $\kappa_2 \leftarrow k_2 - C = 29$.
  - The RNS representation in base $\mathcal{B}$ of κ_2 is $k_2^{(0)} = 7$, $k_2^{(1)} = 5$.
  - The computation steps 11-12 gives $C \leftarrow 2$, and

$$\kappa_2 \leftarrow (8, 5).$$

Values returned by the algorithm:

$$\kappa = ((5,0), (2,6), (8,5), \quad), \text{ and}$$

Recoding Algorithm Example

Let us take $k = 936192_{10}$ (exponent size t of 20 bits, $0 < k < 2^{20}$)

$$k = 48 + 78 \cdot 88 + 32 \cdot 88^2 + 1 \cdot 88^3.$$

Values returned by the algorithm:

$$\kappa = ((5,0),(2,6),(8,5),(3,7)), \text{ and } C = -2.$$

Outline

Exponentiation Algorithm

We use this recoding in modular exponentiation as follows:

$$\begin{aligned} g^k \bmod p &= g^{\sum_{i=0}^{\ell-1} k_i \cdot R^i} \bmod p \\ &= g^{\sum_{i=0}^{\ell-1} \kappa_i \cdot R^i} \cdot g^{C \cdot R^\ell} \bmod p \\ &= g^{C \cdot R^\ell} \cdot \prod_{i=0}^{\ell-1} g^{\kappa_i \cdot R^i} \bmod p \end{aligned}$$

with, when $k_1^{(1)} \neq 0$: $g^{\kappa_i \cdot R^i} \bmod p = g^{k_1^{(1)} \cdot R^i \cdot |k_i' \cdot m_0' + m_1'|_R} \bmod p$,

and, when $k_1^{(1)} = 0$: $g^{\kappa_i \cdot R^i} \bmod p = g^{R^i \cdot ||k_i^{(0)}+1|_{m_0} \cdot m_0' + m_1'|_R} \cdot g^{-R^i \cdot |m_0' + m_1'|_R} \bmod p$.

Exponentiation Algorithm

We use this recoding in modular exponentiation as follows:

$$
\begin{aligned}
g^k \bmod p &= g^{\sum_{i=0}^{\ell-1} k_i \cdot R^i} \bmod p \\
&= g^{\sum_{i=0}^{\ell-1} \kappa_i \cdot R^i} \cdot g^{C \cdot R^\ell} \bmod p \\
&= g^{C \cdot R^\ell} \cdot \prod_{i=0}^{\ell-1} g^{\kappa_i \cdot R^i} \bmod p
\end{aligned}
$$

with, when $k_1^{(1)} \neq 0$: $g^{\kappa_i \cdot R^i} \bmod p = g^{k_1^{(1)} \cdot R^i \cdot |k'_i \cdot m'_0 + m'_1|_R} \bmod p,$

and, when $k_1^{(1)} = 0$: $g^{\kappa_i \cdot R^i} \bmod p = g^{R^i \cdot ||k_i^{(0)} + 1|_{m_0} \cdot m'_0 + m'_1|_R} \cdot g^{-R^i \cdot |m'_0 + m'_1|_R} \bmod p.$

one stores the following values:

$$G_{i,j} = g^{R^i \cdot |j \cdot m'_0 + m'_1|_R} \bmod p, \text{ with } 0 \leq i \leq \ell - 1, 0 \leq j < m_0$$

$$\text{and } G_{\ell,1} = g^{R^\ell \cdot |m'_0 + m'_1|_R} \bmod p.$$

one also stores the following inverses:

$$G_{i,-1} = g^{-R^i \cdot |m'_0 + m'_1|_R} \bmod p \text{ avec } 0 \leq i \leq \ell$$

Exponentiation Algorithm

One uses one value K_j per possible values of $1 \leq \kappa_i^{(1)} < m_1$, that is m_1 points:

$$K_j = \left(\prod_{\text{for all } \kappa_i^{(1)} = j} G_{i,\kappa_i^{(0)}} \right) \times \left(G_{\ell, sign(C)1} \right)_{|C|=j} \bmod p$$

and

$$K_0 = \prod_{\text{for all } \kappa_i^{(1)} = 0} G_{i,\kappa_i^{(0)}} \times G_{i,-1} \bmod p.$$

This leads to

$$g^k \bmod p = K_0 \times \prod_{j=1}^{m_1} K_j^j.$$

Exponentiation Algorithm

Fixed-base $m_0 m_1$ method modular exponentiation

Require: $\{m_0, m_1\}$ RNS base with $R = m_0 m_1$, $k = \sum_{i=0}^{\ell-1} k_i R^i$ and $\kappa = \{\kappa_i, 0 \le i < \ell, (C)\}$ the $m_0 m_1$ recoding of k, p, the DSA modulus, $g \in \mathbb{Z}/p\mathbb{Z}$, public generator of order q.

Ensure: $X = g^k \mod p$

Precomputation. Store $G_{i,j} \leftarrow g^{R^i \cdot |j \cdot m_0' + m_1'|_R}$ with $0 \le i < \ell - 1, 0 \le j < m_0$, $G_{\ell,1} \leftarrow g^{R^\ell \cdot |m_0' + m_1'|_R}$, $G_{i,-1} \leftarrow g^{-R^i \cdot |m_0' + m_1'|_R}, 0 \le i \le \ell$

Computation of the $K_j, 0 \le j < m_1$

$A \leftarrow 1$, $K_j \leftarrow 1$ for $0 \le j < m_1$
for i from 0 to $\ell - 1$ do
    if $\kappa_i^{(1)} = 0$ then
        $K_0 \leftarrow K_0 \times G_{i,\kappa_i^{(0)}} \times G_{i,-1}$
    else
        $K_{\kappa_i^{(1)}} \leftarrow K_{\kappa_i^{(1)}} \times G_{i,\kappa_i^{(0)}}$
    end if
end for
$K_{|C|} \leftarrow K_{|C|} \times G_{\ell, sign(C)1}$

Final Reconstruction

$W \leftarrow$ size of m_1 in bits
for i from W downto 0 do
    $A \leftarrow A^2$
    for j from $m_1 - 1$ downto 1 do
        if bit i of j is non zero then
            $A \leftarrow A \times K_j$
        end if
    end for
end for
return $(A \times K_0)$

Exponentiation Algorithm

Fixed-base $m_0 m_1$ method modular exponentiation

Require: $\{m_0, m_1\}$ RNS base with $R = m_0 m_1$, $k = \sum_{i=0}^{\ell-1} k_i R^i$ and $\kappa = \{\kappa_i, 0 \le i < \ell, (C)\}$ the $m_0 m_1$ recoding of k, p, the DSA modulus, $g \in \mathbb{Z}/p\mathbb{Z}$, public generator of order q.

Ensure: $X = g^k \mod p$

Precomputation. Store $G_{i,j} \leftarrow g^{R^i \cdot |j \cdot m_0' + m_1'|_R}$ with $0 \le i < \ell - 1, 0 \le j < m_0$, $G_{\ell,1} \leftarrow g^{R^\ell \cdot |m_0' + m_1'|_R}$, $G_{i,-1} \leftarrow g^{-R^i \cdot |m_0' + m_1'|_R}$, $0 \le i \le \ell$

TOTAL STORAGE : $(m_0 + 1) \times \ell + m_1 + 2$ elements of $\mathbb{Z}/p\mathbb{Z}$

Computation of the $K_j, 0 \le j < m_1$

$A \leftarrow 1$, $K_j \leftarrow 1$ for $0 \le j < m_1$
for i from 0 to $\ell - 1$ do
    if $\kappa_i^{(1)} = 0$ then
        $K_0 \leftarrow K_0 \times G_{i,\kappa_i^{(0)}} \times G_{i,-1}$
    else
        $K_{\kappa_i^{(1)}} \leftarrow K_{\kappa_i^{(1)}} \times G_{i,\kappa_i^{(0)}}$
    end if
end for
$K_{|C|} \leftarrow K_{|C|} \times G_{\ell, sign(C)1}$

Final Reconstruction

$W \leftarrow$ size of m_1 in bits
for i from W downto 0 do
    $A \leftarrow A^2$
    for j from $m_1 - 1$ downto 1 do
        if bit i of j is non zero then
            $A \leftarrow A \times K_j$
        end if
    end for
end for
return $(A \times K_0)$

Exponentiation Algorithm

Fixed-base $m_0 m_1$ method modular exponentiation

Require: $\{m_0, m_1\}$ RNS base with $R = m_0 m_1$, $k = \sum_{i=0}^{\ell-1} k_i R^i$ and $\kappa = \{\kappa_i, 0 \le i < \ell, (C)\}$ the $m_0 m_1$ recoding of k, p, the DSA modulus, $g \in \mathbb{Z}/p\mathbb{Z}$, public generator of order q.

Ensure: $X = g^k \mod p$

Precomputation. Store $G_{i,j} \leftarrow g^{R^i \cdot |j \cdot m_0' + m_1'|_R}$ with $0 \le i < \ell - 1, 0 \le j < m_0$, $G_{\ell,1} \leftarrow g^{R^\ell \cdot |m_0' + m_1'|_R}$, $G_{i,-1} \leftarrow g^{-R^i \cdot |m_0' + m_1'|_R}$, $0 \le i \le \ell$

TOTAL STORAGE : $(m_0 + 1) \times \ell + m_1 + 2$ elements of $\mathbb{Z}/p\mathbb{Z}$

Computation of the $K_j, 0 \le j < m_1$

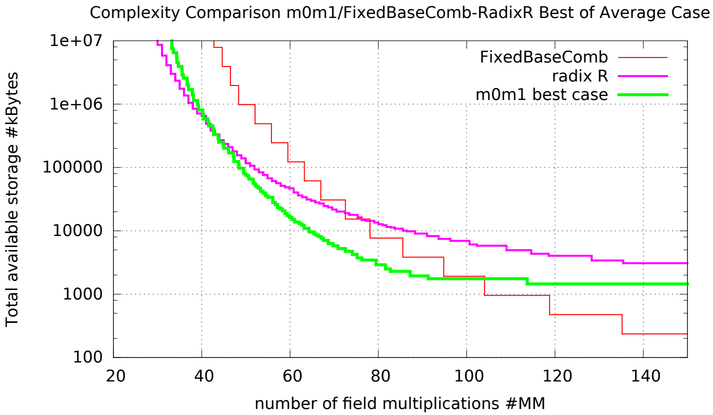
$A \leftarrow 1$, $K_j \leftarrow 1$ for $0 \le j < m_1$
for i from 0 to $\ell - 1$ do
  if $\kappa_i^{(1)} = 0$ then
    $K_0 \leftarrow K_0 \times G_{i,\kappa_i^{(0)}} \times G_{i,-1}$
  else
    $K_{\kappa_i^{(1)}} \leftarrow K_{\kappa_i^{(1)}} \times G_{i,\kappa_i^{(0)}}$
  end if
end for
$K_{|C|} \leftarrow K_{|C|} \times G_{\ell,sign(C)1}$

Final Reconstruction

$W \leftarrow$ size of m_1 in bits
for i from W downto 0 do
  $A \leftarrow A^2$
  for j from $m_1 - 1$ downto 1 do
    if bit i of j is non zero then
      $A \leftarrow A \times K_j$
    end if
  end for
end for
return $(A \times K_0)$

Complexity : $(\ell \frac{m_1+1}{m_1} - m_1)$ MM $+\mathcal{H}$ MM $+(W-1)$ MS

Complexity of the Exponentiation Algorithm



key size $t = 512$ bits (MS $= 0.86 \times$ MM).

$m_0 m_1$ Exponentiation Algorithm Example

Our previous example:

- $k = 936192_{10}$, exponent size t of 20 bits, and $\mathcal{B} = \{11, 8\}$, (i.e. $m_0 = 11, m_1 = 8$);
- radix $R = m_0 \cdot m_1 = 88$ ($\ell = 4$);

$$\kappa = ((5,0),(2,6),(8,5),(3,7)), \text{ and } C = -2.$$

$m_0 m_1$ Exponentiation Algorithm Example

$k = 936192_{10}, \kappa = ((5,0),(2,6),(8,5),(3,7))$, and $C = -2$.

In terms of storage, one computes the values

$$G_{i,j} = g^{R^i \cdot |j \cdot m_0' + m_1'|_R} \mod p \text{ with } 0 \le i \le \ell - 1.$$

One has the following values of $\left|j \cdot m_0' + m_1'\right|_R$ for $0 \le j < 11$:

$$\{33, 1, 57, 25, 81, 49, 17, 73, 41, 9, 65\}$$

In our case, with the chosen parameters, this brings us to store the following values in $\mathbb{Z}/p\mathbb{Z}$:

$$G_i = \{g^{88^i \cdot 33}, g^{88^i}, g^{88^i \cdot 57}, g^{88^i \cdot 25}, g^{88^i \cdot 81}, g^{88^i \cdot 49}, g^{88^i \cdot 17}, g^{88^i \cdot 73}, g^{88^i \cdot 41}, g^{88^i \cdot 9}, g^{88^i \cdot 65}\}.$$

m_0m_1 Exponentiation Algorithm Example

$$k = 936192_{10}, \kappa = ((5,0),(2,6),(8,5),(3,7)), \text{ and } C = -2.$$

$$G_i = \{g^{88^i \cdot 33}, g^{88^i}, g^{88^i \cdot 57}, g^{88^i \cdot 25}, g^{88^i \cdot 81}, g^{88^i \cdot 49}, g^{88^i \cdot 17}, g^{88^i \cdot 73}, g^{88^i \cdot 41}, g^{88^i \cdot 9}, g^{88^i \cdot 65}\}.$$

First iteration ($i = 0$), $\kappa_0^{(1)} = 0$ (and $\kappa_0^{(0)} = 5$), and this gives

$$K_0 \leftarrow G_{0,\kappa_0^{(0)}} \times G_{0,-1} = g^{49} \times g^{-1} = g^{48}.$$

$K_0 \leftarrow g^{48}$,

m_0m_1 Exponentiation Algorithm Example

$k = 936192_{10}, \kappa = ((5,0), (2,6), (8,5), (3,7))$, and $C = -2$.

$$G_i = \{g^{88^i \cdot 33}, g^{88^i}, g^{88^i \cdot 57}, g^{88^i \cdot 25}, g^{88^i \cdot 81}, g^{88^i \cdot 49}, g^{88^i \cdot 17}, g^{88^i \cdot 73}, g^{88^i \cdot 41}, g^{88^i \cdot 9}, g^{88^i \cdot 65}\}.$$

Second iteration ($i = 1$), $\kappa_1^{(1)} = 6$ (and $\kappa_1^{(0)} = 2$), and this gives

$$K_6 \leftarrow G_{1,\kappa_1^{(0)}} = g^{88 \cdot 57} = g^{5016}.$$

$K_0 \leftarrow g^{48}$, $K_6 \leftarrow g^{5016}$,

$m_0 m_1$ Exponentiation Algorithm Example

$k = 936192_{10}, \kappa = ((5,0),(2,6),(8,5),(3,7))$, and $C = -2$.

$$G_i = \{g^{88^i \cdot 33}, g^{88^i}, g^{88^i \cdot 57}, g^{88^i \cdot 25}, g^{88^i \cdot 81}, g^{88^i \cdot 49}, g^{88^i \cdot 17}, g^{88^i \cdot 73}, g^{88^i \cdot 41}, g^{88^i \cdot 9}, g^{88^i \cdot 65}\}.$$

Third iteration ($i = 2$), $\kappa_2^{(1)} = 5$ (and $\kappa_2^{(0)} = 8$), and this gives

$$K_5 \leftarrow G_{2,\kappa_2^{(0)}} = g^{88^2 \cdot 41} = g^{317504}.$$

$K_0 \leftarrow g^{48}$, $K_5 \leftarrow g^{317504}$, $K_6 \leftarrow g^{5016}$,

$m_0 m_1$ Exponentiation Algorithm Example

$k = 936192_{10}, \kappa = ((5,0),(2,6),(8,5),(3,7))$, and $C = -2$.

$$G_i = \{g^{88^i \cdot 33}, g^{88^i}, g^{88^i \cdot 57}, g^{88^i \cdot 25}, g^{88^i \cdot 81}, g^{88^i \cdot 49}, g^{88^i \cdot 17}, g^{88^i \cdot 73}, g^{88^i \cdot 41}, g^{88^i \cdot 9}, g^{88^i \cdot 65}\}.$$

Fourth iteration ($i = 3$), $\kappa_2^{(1)} = 7$ (and $\kappa_2^{(0)} = 3$), and this gives

$$K_7 \leftarrow G_{3,\kappa_2^{(0)}} = g^{88^3 \cdot 25} = g^{17036800}.$$

$K_0 \leftarrow g^{48}$, $K_5 \leftarrow g^{317504}$, $K_6 \leftarrow g^{5016}$, $K_7 \leftarrow g^{17036800}$.

$m_0 m_1$ Exponentiation Algorithm Example

$k = 936192_{10}, \kappa = ((5,0),(2,6),(8,5),(3,7))$, and $C = -2$.

$$G_i = \{g^{88^i \cdot 33}, g^{88^i}, g^{88^i \cdot 57}, g^{88^i \cdot 25}, g^{88^i \cdot 81}, g^{88^i \cdot 49}, g^{88^i \cdot 17}, g^{88^i \cdot 73}, g^{88^i \cdot 41}, g^{88^i \cdot 9}, g^{88^i \cdot 65}\}.$$

The last carry $C = -2$ is now processed:

$$K_2 \leftarrow G_{4,-1} = g^{88^4 \cdot (-1)} = g^{-59969536}.$$

$$K_0 \leftarrow g^{48}, \quad K_2 \leftarrow g^{-59969536}, \quad K_5 \leftarrow g^{317504}, \quad K_6 \leftarrow g^{5016}, \quad K_7 \leftarrow g^{17036800}.$$

$m_0 m_1$ Exponentiation Algorithm Example

$k = 936192_{10}, \kappa = ((5,0),(2,6),(8,5),(3,7))$, and $C = -2$.

$$G_i = \{g^{88^i \cdot 33}, g^{88^i}, g^{88^i \cdot 57}, g^{88^i \cdot 25}, g^{88^i \cdot 81}, g^{88^i \cdot 49}, g^{88^i \cdot 17}, g^{88^i \cdot 73}, g^{88^i \cdot 41}, g^{88^i \cdot 9}, g^{88^i \cdot 65}\}.$$

Final Reconstruction

$$\begin{aligned} g^k \bmod p &= K_0 \times \prod_{j=1}^{m_1} K_j^j \bmod p \\ &= g^{48+2\cdot(-59969536)+5\cdot317504+6\cdot5016+7\cdot17036800} \bmod p \\ &= g^{936192} \bmod p, \end{aligned}$$

$\rightarrow$ which is the desired result.

$$K_0 \leftarrow g^{48}, \quad K_2 \leftarrow g^{-59969536}, \quad K_5 \leftarrow g^{317504}, \quad K_6 \leftarrow g^{5016}, \quad K_7 \leftarrow g^{17036800}.$$

Outline

Implementation of the m_0m_1 exponentiation algorithm

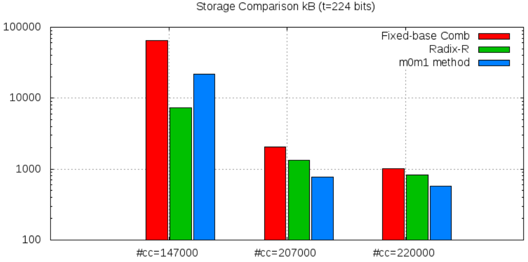
For the three considered exponentiation algorithms:

- C language, compiled with `gcc 4.8.3`;
- Platform: CPU Intel XEON® E5-2650 (Ivy bridge), CENTOS 7.0.1406, RAM 12.6 GBytes;
- Multiprecision Multiplication and Squaring: `GMP` library;
- Modular Reduction: block Montgomery approach;
- m_0m_1 Recoding: `GMP` library;
- Test processing : a few hundred of dataset for each size, with multiple run and averaging of the minimum of every dataset;
- The timings in clock cycles includes the recoding;
- Tests for the following standards (fips 186-4):

NIST key size (bits)	224	256	384	512
field size (bits)	2048	3072	7680	15360

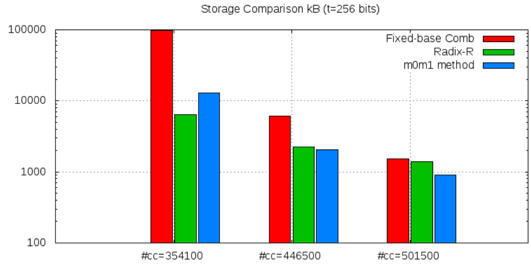
Performances

Modular Exponentiation			
State of the Art methods: *Fixed-base Comb*	State of the Art methods: radix R	m_0, m_1 rec.	ratio
$\#CC$ Storage	$\#CC$ Storage	$\#CC$ Storage	m_0, m_1 /Best S.o.A.
key size 224 bits, field size 2048 bits (level of security: 112 bits)			
221108 1023.5 kB ($w = 12$)	227838 829 kB ($R = 91$)	**219864** **580 kB** ($m_0 = 89, m_1 = 6$)	$\times 0.994$ $\times 0.700$
210074 2047.5 kB ($w = 13$)	206888 1324 kB ($R = 163$)	**207072** **766 kB** ($m_0 = 127, m_1 = 7$)	$\times 0.985$ $\times 0.579$
149690 65535 kB ($w = 18$)	147877 **7289kB** ($R = 1223$)	**146156** 21599 kB ($m_0 = 5417, m_1 = 6$)	$\times 0.988$ $\times 2.96$



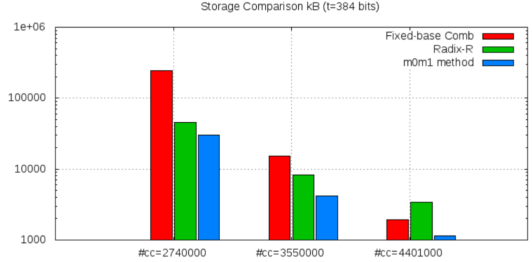
Performances

Modular Exponentiation			
State of the Art methods			
Fixed-base Comb	radix R	m_0, m_1 rec.	ratio
$\#CC$ Storage	$\#CC$ Storage	$\#CC$ Storage	m_0, m_1 /Best S.o.A.
key size 256 bits, field size 3072 bits (level of security: 128 bits)			
524539 1535 kB ($w = 12$)	502981 1411 kB ($R = 91$)	**501466** **897 kB ($m_0 = 79, m_1 = 6$)**	$\times$0.997 $\times$0.636
449397 6143 kB ($w = 14$)	445871 2251 kB ($R = 163$)	**446444** **2056 kB ($m_0 = 211, m_1 = 6$)**	$\times$1.001 $\times$0.913
356892 98303 kB ($w = 18$)	354640 **6414 kB ($R = 571$)**	354071 12843 kB ($m_0 = 1721, m_1 = 7$)	$\times$0.998 $\times$2.002



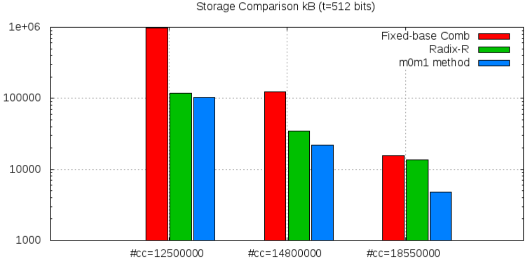
Performances

Modular Exponentiation			
State of the Art methods			
Fixed-base Comb	radix R	m_0, m_1 rec.	ratio
$\#CC$ Storage	$\#CC$ Storage	$\#CC$ Storage	m_0, m_1 /Best S.o.A.
key size 384 bits, field size 7680 bits (level of security: 192 bits)			
4442590 1918 kB ($w = 11$)	4492191 3430 kB ($R = 53$)	**4409584** **1134 kB ($m_0 = 23$, $m_1 = 10$)**	$\times$0.993 $\times$0.467
3554339 15358 kB ($w = 14$)	**3524896** 8290 kB ($R = 163$)	3551437 **4164 kB ($m_0 = 113$, $m_1 = 10$)**	$\times$1.008 $\times$0.502
2736341 245758 kB ($w = 18$)	**2543480** 45221 kB ($R = 1223$)	2743399 **29961 kB ($m_0 = 1031$, $m_1 = 7$)**	$\times$1.079 $\times$0.662



Performances

Modular Exponentiation			
State of the Art methods			
Fixed-base Comb	radix R	m_0, m_1 rec.	ratio
$\#CC$ Storage	$\#CC$ Storage	$\#CC$ Storage	m_0, m_1 /Best S.o.A.
key size 512 bits, field size 15360 bits (level of security: 256 bits)			
18632429 15536 kB ($w = 13$)	19260731 13765 kB ($R = 91$)	**18550238** **4745 kB ($m_0 = 41, m_1 = 10$)**	$\times 0.996$ $\times 0.345$
14848261 122876 kB ($w = 16$)	15401002 34418 kB ($R = 163$)	**14813453** **22109 kB ($m_0 = 257, m_1 = 11$)**	$\times 0.998$ $\times 0.642$
12477816 983036 kB ($w = 19$)	**12193232** 119061 kB ($R = 1223$)	12499600 **102820 kB ($m_0 = 1381, m_1 = 7$)**	$\times 1.025$ $\times 0.863$



Outline

Conclusion and future work

We have presented:

- DSA signature protocol;
- Main State of the Art approaches for modular exponentiation;
- Our Contributions:
  - RNS digit recoding for exponent;
  - Enhanced algorithms for modular exponentiation;
  - Software implementations;
  - Performance results: storage saving up to three times;

Conclusion and future work

We have presented:

- DSA signature protocol;
- Main State of the Art approaches for modular exponentiation;
- Our Contributions:
  - RNS digit recoding for exponent;
  - Enhanced algorithms for modular exponentiation;
  - Software implementations;
  - Performance results: storage saving up to three times;

Future work:

- Application to elliptic curve signature (ECDSA);
- Develop improvements to thwart side-channel analysis (cache attack...);
- Hardware and embedded systems implementations.

Thank you for your attention,

Any questions ?

mail : jean-marc.robert@univ-perp.fr

home page : http://perso.univ-perp.fr/jeanmarc.robert

Work funded by ANR PAVOIS 12 BS02 002 02 project