Enhanced Digital Signature using Splitted Exponent Digit Representation

Christophe Nègre[1], Thomas Plantard[2], Jean-Marc Robert[1,3]

1: Team DALI/LIRMM, University of Perpignan, France
2: CCISR, SCIT, University of Wollongong, Australia
3: IMATH, Université de Toulon

le 18 avril 2019

WRACH 2019, Roscoff, France

Institut de mathématiques de Toulon
Université de Toulon

UNIVERSITÉ DE TOULON

Table des matières

Table des matières

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 ← 1
for i from t − 1 downto 0 do
    X ← X^2 mod p
    if k_i = 1 then
        X ← X · g mod 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 ← 1
for i from t − 1 downto 0 do
    X ← X^2 mod p
    if k_i = 1 then
        X ← X · g mod p
    end if
end for
return (X)
```

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

$\Rightarrow$ One takes no advantage of the reuse of the exponent
(i.e. when one needs to compute a lot of signature with the same public key)

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 $G_{i,j} \leftarrow g^{j \cdot R^i}$, with $j \in [1, ..., R-1]$ and $0 \leq i < \ell$.
$X \leftarrow 1$
for i from $\ell - 1$ downto 0 **do**
  $X \leftarrow X \cdot G_{i,k_i} \bmod p$
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 $G_{i,j} \leftarrow g^{j \cdot R^i}$, with $j \in [1, ..., R-1]$ and $0 \leq i < \ell$.
$X \leftarrow 1$
for i from $\ell - 1$ downto 0 **do**
  $X \leftarrow X \cdot G_{i,k_i} \bmod p$
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

In this method, the exponent k is written in w rows, and the colums are processed one at a time. Thus, $d = \lceil t/w \rceil$ is the column size.

$$k = K^{w-1} \| \ldots \| K^1 \| K^0$$

Each K^j is a bit string of length d. Let K_i^j denote the i^{th} bit of K^j.
One sets: $g^{[K_i^{w-1},\ldots,K_i^1,K_i^0]} = g^{K_i^{w-1}2^{(w-1)d}+\ldots+K_i^2 2^{2d}+K_i^1 2^d+K_i^0}$

Fixed-base Comb Method

One sets: $g^{[K_i^{w-1},\ldots,K_i^1,K_i^0]} = g^{K_i^{w-1}2^{(w-1)d}+\ldots+K_i^2 2^{2d}+K_i^1 2^d+K_i^0}$

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: $X = g^k \bmod p$
Precomputation. Compute and store $g^{[a_{w-1},\ldots,a_0]} \bmod p$, $\forall (a_{w-1},\ldots,a_0) \in \mathbb{Z}_2^w$.
$X \leftarrow 1$
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

One sets: $g^{[K_i^{w-1},\dots,K_i^1,K_i^0]} = g^{K_i^{w-1}2^{(w-1)d}+\dots+K_i^2 2^{2d}+K_i^1 2^d+K_i^0}$

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

Require: $k = (k_{t-1},\dots,k_1,k_0)_2$, 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: $X = g^k \bmod p$
Precomputation. Compute and store $g^{[a_{w-1},\dots,a_0]} \bmod p$, $\forall (a_{w-1},\dots,a_0) \in \mathbb{Z}_2^w$.
$X \leftarrow 1$
for i from $d-1$ downto 0 **do**
  $X \leftarrow X^2 \bmod p$
  $X \leftarrow X \cdot g^{[K_i^{w-1},\dots,K_i^1,K_i^0]} \bmod p$
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

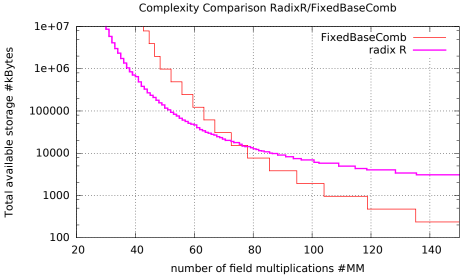
Complexities and storage amounts of state of the art methods, average case.

	# 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).

Table des matières

Contributions

Starting from the Radix-R method:

- Digit recoding for exponent, using a multiplicative splitting (2 approaches);

Contributions

Starting from the Radix-R method:

- Digit recoding for exponent, using a multiplicative splitting (2 approaches);
- Enhanced algorithm for Modular Exponentiation and Elliptic Curve Scalar Multiplication;

Contributions

Starting from the Radix-R method:

- Digit recoding for exponent, using a multiplicative splitting (2 approaches);
- Enhanced algorithm for Modular Exponentiation and Elliptic Curve Scalar Multiplication;
- Complexity and storage requirements evaluation;

Contributions

Starting from the Radix-R method:

- Digit recoding for exponent, using a multiplicative splitting (2 approaches);
- Enhanced algorithm for Modular Exponentiation and Elliptic Curve Scalar Multiplication;
- Complexity and storage requirements evaluation;
- Software implementations, showing performance improvements.

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 → RNS splitting

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 → RNS splitting

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 → RNS splitting

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

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

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 → RNS splitting

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

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

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 $k_{i+1} \geq 0$.

General Idea for Modular Exponentiation → RNS splitting

Radix-R method:

Stores $G_{i,j} \leftarrow g^{j \cdot R^i}$, $(0 \leq j < R)$

Computes $\prod_{i=0}^{\ell-1} G_{i,k_i}$.

$\Rightarrow$ $\approx$ Low complexity, large storage

$\Rightarrow$

Variant:

Stores $G_i \leftarrow g^{R^i}$;

Computes:

$$\prod_{j=0}^{R-1} \left(\prod_{\forall i, 0 \leq i < \ell-1, k_i = j} G_i \right)^j.$$

$\Rightarrow$ $\approx$ Low storage, large complexity

General Idea for Modular Exponentiation → RNS splitting

Radix-R method:

Stores $G_{i,j} \leftarrow g^{j \cdot R^i}$, $(0 \leq j < R)$

Computes $\prod_{i=0}^{\ell-1} G_{i,k_i}$.

$\Rightarrow$ $\approx$ Low complexity, large storage

Variant:

Stores $G_i \leftarrow g^{R^i}$;

Computes:

$\prod_{j=0}^{R-1} \left(\prod_{\forall i, 0 \leq i < \ell-1, k_i = j} G_i \right)^j$.

$\Rightarrow$ $\approx$ Low storage, large complexity

$\Rightarrow$

Our method ($m_0 m_1$ RNS):

Stores $G_{i,\tilde{j}} \leftarrow g^{f(\tilde{j}) \cdot R^i}$, $(0 \leq \tilde{j} < m_0)$

Computes $K_0 \times \prod_{i=1}^{m_1} K_i^i$ with

$K_i = \prod_{j=1, \tilde{k}_j^{(1)} = i}^{\ell-1} G_{i, \tilde{k}_j^{(0)}}^{\tilde{k}_j^{(1)}}$

$\Rightarrow$ $\approx$ Better trade-off.

Exponentiation Algorithm → RNS splitting

Fixed-base $m_0 m_1$ method modular exponentiation

Require: $k = \sum_{i=0}^{\ell-1} k_i R^i$ and $\kappa = \{\kappa_i, 0 \le i < \ell, (C)\}$ the $m_0 m_1$ recoding of k.
Ensure: $X = g^k \mod p$

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

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 $k_i^{(1)} = 0$ **then**
    $K_0 \leftarrow K_0 \times G_{i,(k_i^{(0)}+1)} \times G_{i,-1}$
  else
    $K_{k_i^{(1)}} \leftarrow K_{k_i^{(1)}} \times G_{i,k_i'^{(0)}}$
  end if
end for
$K_{|C|} \leftarrow K_{|C|} \times G_{\ell,sign(C)1}$

Final Reconstruction

return $(K_0 \times \prod_{j=1}^{m_1} K_j^j)$

Exponentiation Algorithm → RNS splitting

Fixed-base $m_0 m_1$ method modular exponentiation

Require: $k = \sum_{i=0}^{\ell-1} k_i R^i$ and $\kappa = \{\kappa_i, 0 \le i < \ell, (C)\}$ the $m_0 m_1$ recoding of k.
Ensure: $X = g^k \mod p$

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

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 $k_i^{(1)} = 0$ **then**
    $K_0 \leftarrow K_0 \times G_{i,(k_i^{(0)}+1)} \times G_{i,-1}$
  else
    $K_{k_i^{(1)}} \leftarrow K_{k_i^{(1)}} \times G_{i,k_i'^{(0)}}$
  end if
end for
$K_{|C|} \leftarrow K_{|C|} \times G_{\ell,sign(C)1}$

Final Reconstruction

return $(K_0 \times \prod_{j=1}^{m_1} K_j^j)$

Exponentiation Algorithm → RNS splitting

Fixed-base $m_0 m_1$ method modular exponentiation

Require: $k = \sum_{i=0}^{\ell-1} k_i R^i$ and $\kappa = \{\kappa_i, 0 \leq i < \ell, (C)\}$ the $m_0 m_1$ recoding of k.
Ensure: $X = g^k \mod p$

Precomputation. Store $G_{i,j} \leftarrow g^{R^i \cdot |j \cdot m'_0 + m'_1|_R}$, $G_{\ell,1} \leftarrow g^{R^\ell \cdot |m'_0 + m'_1|_R}$, $G_{i,-1} \leftarrow g^{-R^i \cdot |m'_0 + m'_1|_R}$

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

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

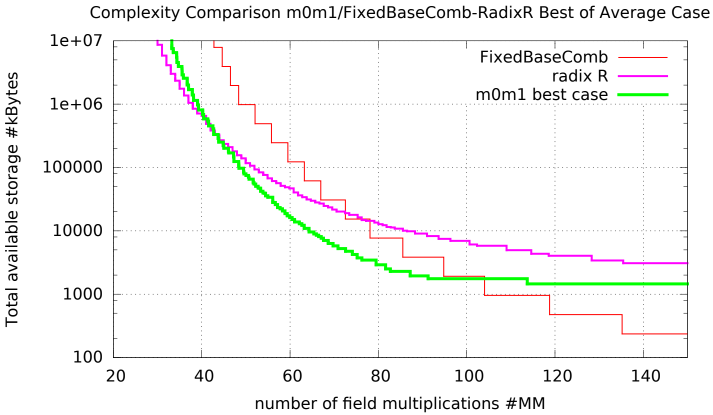
$A \leftarrow 1, K_j \leftarrow 1$ for $0 \leq j < m_1$
for i from 0 to $\ell - 1$ **do**
  if $k_i^{(1)} = 0$ **then**
    $K_0 \leftarrow K_0 \times G_{i,(k_i^{(0)}+1)} \times G_{i,-1}$
  else
    $K_{k_i^{(1)}} \leftarrow K_{k_i^{(1)}} \times G_{i,k_i'^{(0)}}$
  end if
end for
$K_{|C|} \leftarrow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

Final Reconstruction

return $(K_0 \times \prod_{j=1}^{m_1} K_j^j)$

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).

Application of the $m_0 m_1$ method to Elliptic Curve Cryptography

- Is the $m_0 m_1$ recoding suitable for ECC?

$\Rightarrow$ NO!

The $m_0 m_1$ recoding does not perform better than the S-o-A algorithms in the ECC case : how to devise a suitable recoding?

Application of the m_0m_1 method to Elliptic Curve Cryptography

- Is the m_0m_1 recoding suitable for ECC?

$\Rightarrow$ NO!

The m_0m_1 recoding does not perform better than the S-o-A algorithms in the ECC case : how to devise a suitable recoding?

- Drawback of the m_0m_1 based exponentiation :

not constant time computation (see the algorithm).

Is it possible to improve the algorithm to render it side-channel attack resistant?

Recoding Algorithm $\rightarrow$ R-splitting

k is the scalar, represented in radix R, prime integer:

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

$\Rightarrow$ Extended Euclidean Algorithm:
(EEA, r_j is the sequence of Euclidean remainders):

$$r_j = u_j \times R + v_j \times k_i. \tag{1}$$

One sets c the upper bound of r_j, to terminate the EEA (and $\lceil R/c \rceil$ is the upper bound of $|v_j|$). We then keep $k_i^{(0)} = r_j$ and $k_i^{(1)} = v_j$.

After (1), since R is prime, one stops the EEA such as

$$k_i = |k_i^{(0)} \times (k_i^{(1)})^{-1}|_R, \text{ with } k_i^{(0)} < c \text{ and } |k_i^{(1)}| \leq \lceil R/c \rceil.$$

R-splitting Recoding Algorithm

After (1), since R is prime, one stops the EEA such as

$$k_i = |k_i^{(0)} \times (k_i^{(1)})^{-1}|_R, \text{ with } k_i^{(0)} < c \text{ and } |k_i^{(1)}| \leq \lceil R/c \rceil.$$

Modular reduction mod R: one distinguishes the cases $k_i^{(1)} > 0$ and $k_i^{(1)} < 0$

R-splitting Recoding Algorithm

After (1), since R is prime, one stops the EEA such as

$$k_i = |k_i^{(0)} \times (k_i^{(1)})^{-1}|_R, \text{ with } k_i^{(0)} < c \text{ and } |k_i^{(1)}| \leq \lceil R/c \rceil.$$

Modular reduction mod R: one distinguishes the cases $k_i^{(1)} > 0$ and $k_i^{(1)} < 0$

- if $k_i^{(1)} > 0$, one proceeds as previously:

$$k_i = k_i^{(0)} \cdot |(k_i^{(1)})^{-1}|_R - \overbrace{\left\lfloor \frac{k_i^{(0)} \cdot |(k_i^{(1)})^{-1}|_R}{R} \right\rfloor}^{C} \cdot R.$$

Let us denote $C = \left\lfloor \frac{k_i^{(0)} \cdot |(k_i^{(1)})^{-1}|_R}{R} \right\rfloor$ $(0 \leq C \leq c < R)$

$$\text{if } k_{i+1} \geq C \text{ then } k_{i+1} \leftarrow k_{i+1} - C, C \leftarrow 0,$$

$$\text{else } k_{i+1} \leftarrow k_{i+1} + R - C, C \leftarrow 1.$$

R-splitting Recoding Algorithm

After (1), since R is prime, one stops the EEA such as

$$k_i = |k_i^{(0)} \times (k_i^{(1)})^{-1}|_R, \text{ with } k_i^{(0)} < c \text{ and } |k_i^{(1)}| \leq \lceil R/c \rceil.$$

Modular reduction mod R: one distinguishes the cases $k_i^{(1)} > 0$ and $k_i^{(1)} < 0$

- if $k_i^{(1)} < 0$, one proceeds slightly differently:

$$k_i = k_i^{(0)} \cdot (R - |(-k_i^{(1)})^{-1}|_R) - \overbrace{\left\lfloor \frac{k_i^{(0)} \cdot |(k_i^{(1)})^{-1}|_R}{R} \right\rfloor}^{c} \cdot R.$$

Let us denote $C = \left\lfloor \frac{k_i^{(0)} \cdot |(k_i^{(1)})^{-1}|_R}{R} \right\rfloor - k_i^{(0)}$ $(-c \leq C \leq c < R)$

$$k_{i+1} \leftarrow k_{i+1} - C, C \leftarrow -\lfloor k_{i+1}/R \rfloor, k_{i+1} \leftarrow |k_{i+1}|_R$$

R-splitting Recoding Algorithm

One notices:

- The case $k_i^{(1)} = 0$ does not need to be taken into account;
- it might be necessary to process a last carry C.

$\rightarrow$ The sequence of the $\kappa_i \leftarrow (k_i', k_i^{(1)})$ is the R-splitting recoding of k.

Back to the General Idea for ECC $\rightarrow$ R-splitting

Radix-R method:

Stores $M_{i,j} \leftarrow j \cdot R^i \cdot P$, $(0 \leq j < R)$

Computes $\sum_{i=0}^{\ell-1} M_{i,k_i}$.

$\Rightarrow \quad \approx$ Low complexity, large storage

$\Longrightarrow$

Variant:

Stores $M_i \leftarrow R^i \cdot P$;

Computes:

$$\sum_{j=0}^{R-1} \left(\sum_{\forall i, 0 \leq i < \ell-1, k_i = j} j \cdot M_i \right).$$

$\Rightarrow \quad \approx$ Low storage, large complexity

Back to the General Idea for ECC $\rightarrow$ R-splitting

Radix-R method:

Stores $M_{i,j} \leftarrow j \cdot R^i \cdot P$, $(0 \leq j < R)$

Computes $\sum_{i=0}^{\ell-1} M_{i,k_i}$.

$\Rightarrow$ $\approx$ Low complexity, large storage

Variant:

Stores $M_i \leftarrow R^i \cdot P$;

Computes:

$$\sum_{j=0}^{R-1} \left(\sum_{\forall i, 0 \leq i < \ell-1, k_i = j} j \cdot M_i \right).$$

$\Rightarrow$ $\approx$ Low storage, large complexity

$\Rightarrow$

Our method (R-splitting):

Stores $M_{i,\tilde{j}} \leftarrow f(\tilde{j}) \cdot R^i \cdot P$,
$(0 \leq \tilde{j} < c)$

Computes $\sum_{i=1}^{c} i \cdot K_i$ with

$$K_i = \sum_{j=1, \tilde{k}_j^{(1)}=i}^{\ell-1} \tilde{k}_j^{(1)} \cdot M_{i,\tilde{k}_j^{(0)}}$$

$\Rightarrow$ $\approx$ Better trade-off.

ECSM $\rightarrow$ R-splitting

We can now take into account the Side-channel resistance:

Fixed-base R-splitting method ECSM

Require: A prime integer R ,a scalar $k = \sum_{i=0}^{\ell-1} k_i R^i$ with $= \{(s_i, k_i^{(0)}, k_i^{(1)}), 0 \leq i < \ell, (k'_\ell)\}$ its multiplicative splitting recoding using W-bit split c and a fixed point $P \in E(\mathbb{F}_p)$.
Ensure: $X = k \cdot P$
Precomputation. Store $T[i][j] \leftarrow (\left|j^{-1}\right|_R \cdot R^i) \cdot P$ for $i = 0, \ldots, \ell-1, j = 1, \ldots, \lceil R/c \rceil$ and $T[\ell][1] \leftarrow R^\ell \cdot P$ and $T[i][0] \leftarrow \mathcal{O}$ for $i = 0, \ldots, \ell-1$.

Computation of the $Y_j, 1 \leq j \leq c$

$X \leftarrow \mathcal{O}, Y_j \leftarrow \mathcal{O}$ for $1 \leq j \leq c$
for i from 0 to $\ell - 1$ do
    $Y_{k_i^{(0)}} \leftarrow Y_{k_i^{(0)}} + (s_i) \cdot T[i][k_i^{(1)}]$
end for //regular loop.
$Y_{|k'_\ell|} \leftarrow Y_{|k'_\ell|} + (\text{sign}(k'_\ell)) \cdot T[\ell][1]$

Final Reconstruction

return $(X \leftarrow \sum_{j=1}^{W} j \cdot Y_j)$

ECSM $\rightarrow$ R-splitting

We can now take into account the Side-channel resistance:

Fixed-base R-splitting method ECSM

Require: A prime integer R ,a scalar $k = \sum_{i=0}^{\ell-1} k_i R^i$ with $= \{(s_i, k_i^{(0)}, k_i^{(1)}), 0 \leq i < \ell, (k'_\ell)\}$ its multiplicative splitting recoding using W-bit split c and a fixed point $P \in E(\mathbb{F}_p)$.
Ensure: $X = k \cdot P$
Precomputation. Store $T[i][j] \leftarrow (\left|j^{-1}\right|_R \cdot R^i) \cdot P$ for $i = 0, \ldots, \ell-1, j = 1, \ldots, \lceil R/c \rceil$ and $T[\ell][1] \leftarrow R^\ell \cdot P$ and $T[i][0] \leftarrow \mathcal{O}$ for $i = 0, \ldots, \ell-1$.

Computation of the $Y_j, 1 \leq j \leq c$

$X \leftarrow \mathcal{O}, Y_j \leftarrow \mathcal{O}$ for $1 \leq j \leq c$
for i from 0 to $\ell - 1$ **do**
$Y_{k_i^{(0)}} \leftarrow Y_{k_i^{(0)}} + (s_i) \cdot T[i][k_i^{(1)}]$
end for //regular loop.
$Y_{|k'_\ell|} \leftarrow Y_{|k'_\ell|} + (\text{sign}(k'_\ell)) \cdot T[\ell][1]$

TOTAL STORAGE:
$(\ell \times \lceil R/c \rceil + c)$ EC points

Final Reconstruction

return $(X \leftarrow \sum_{j=1}^{W} j \cdot Y_j)$

ECSM $\rightarrow$ R-splitting

We can now take into account the Side-channel resistance:

Fixed-base R-splitting method ECSM

Require: A prime integer R ,a scalar $k = \sum_{i=0}^{\ell-1} k_i R^i$ with $= \{(s_i, k_i^{(0)}, k_i^{(1)}), 0 \leq i < \ell, (k'_\ell)\}$ its multiplicative splitting recoding using W-bit split c and a fixed point $P \in E(\mathbb{F}_p)$.
Ensure: $X = k \cdot P$
Precomputation. Store $T[i][j] \leftarrow (\left|j^{-1}\right|_R \cdot R^i) \cdot P$ for $i = 0, \ldots, \ell-1, j = 1, \ldots, \lceil R/c \rceil$ and $T[\ell][1] \leftarrow R^\ell \cdot P$ and $T[i][0] \leftarrow \mathcal{O}$ for $i = 0, \ldots, \ell-1$.

Computation of the $Y_j, 1 \leq j \leq c$

$X \leftarrow \mathcal{O}, Y_j \leftarrow \mathcal{O}$ for $1 \leq j \leq c$
for i from 0 to $\ell - 1$ **do**
  $Y_{k_i^{(0)}} \leftarrow Y_{k_i^{(0)}} + (s_i) \cdot T[i][k_i^{(1)}]$
end for //regular loop.
$Y_{|k'_\ell|} \leftarrow Y_{|k'_\ell|} + (\mathrm{sign}(k'_\ell)) \cdot T[\ell][1]$

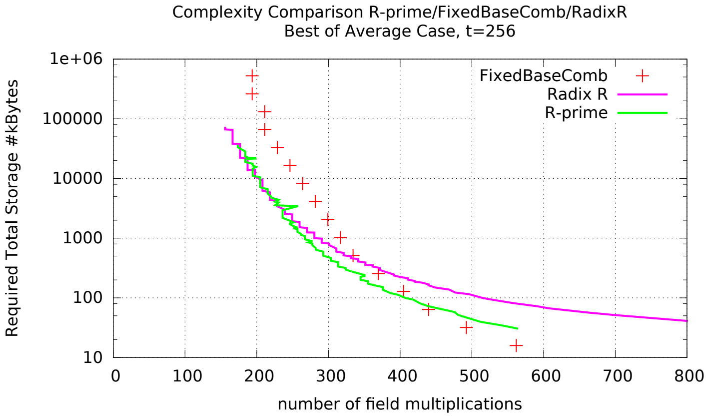
TOTAL STORAGE:
$(\ell \times \lceil R/c \rceil + c)$ EC points

Final Reconstruction

return $(X \leftarrow \sum_{j=1}^{W} j \cdot Y_j)$

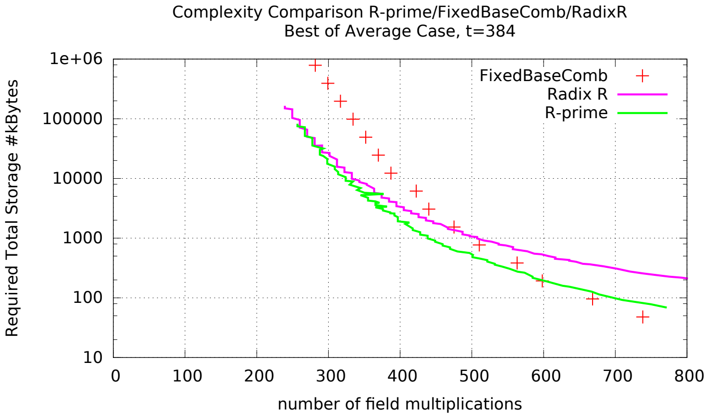
Complexity : $\ell \times MixedAdd + (W-1) \times Dbl + \mathcal{H} \times Add$

Complexity of the ECSM Algorithm → R-splitting



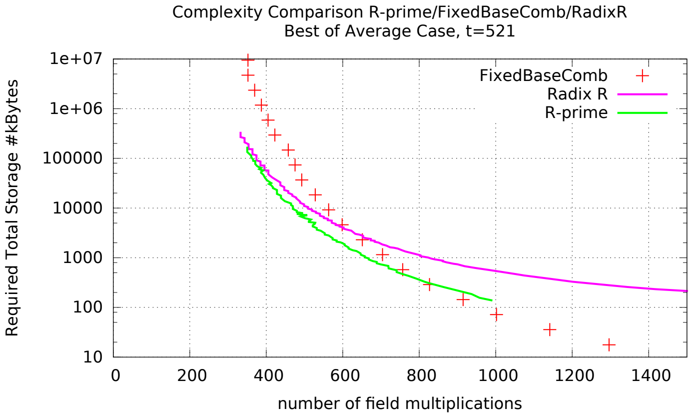
Scalar size $t = 256$ bits.

Complexity of the ECSM Algorithm $\rightarrow$ R-splitting



Scalar size $t = 384$ bits.

Complexity of the ECSM Algorithm → R-splitting



Scalar size $t = 521$ bits.

Implementation of the $m_0 m_1$ exponentiation algorithm

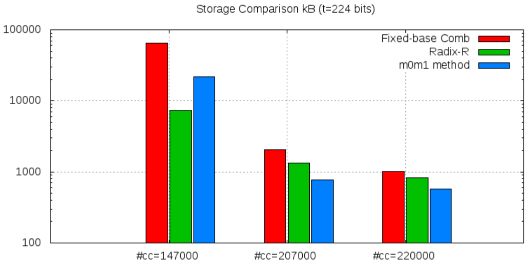
For the three considered exponentiation algorithms:

- C language, compiled with `gcc 4.8.3`;
- Multiprecision Integer Operations: low-level functions of the `GMP` library;
- Modular Reduction: block Montgomery approach;
- 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 include the recoding;
- Tests for the following standards (fips 186-4):

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

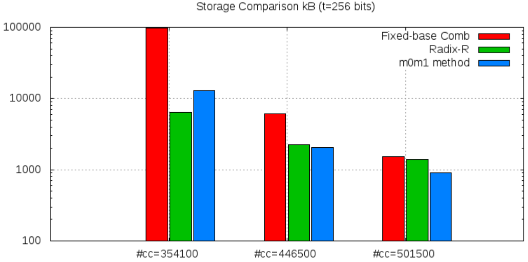
Performances

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 224 bits, field size 2048 bits (level of security: 112 bits)			
221108 CC 1023.5 kB ($w = 12$)	227838 CC 829 kB ($R = 91$)	**219864 CC** **580 kB** ($m_0 = 89, m_1 = 6$)	$\times$0.994 $\times$0.700
210074 CC 2047.5 kB ($w = 13$)	206888 CC 1324 kB ($R = 163$)	**207072 CC** **766 kB** ($m_0 = 127, m_1 = 7$)	$\times$0.985 $\times$0.579
149690 CC 65535 kB ($w = 18$)	147877 CC **7289kB** ($R = 1223$)	**146156 CC** 21599 kB ($m_0 = 5417, m_1 = 6$)	$\times$0.988 $\times$2.96



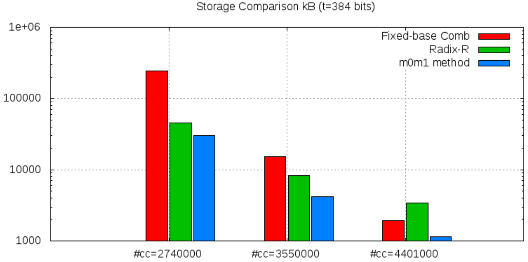
Performances

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 CC 1535 kB ($w = 12$)	502981 CC 1411 kB ($R = 91$)	**501466 CC** **897 kB** ($m_0 = 79$, $m_1 = 6$)	$\times 0.997$ $\times 0.636$
449397 CC 6143 kB ($w = 14$)	445871 CC 2251 kB ($R = 163$)	**446444 CC** **2056 kB** ($m_0 = 211$, $m_1 = 6$)	$\times 1.001$ $\times 0.913$
356892 CC 98303 kB ($w = 18$)	354640 CC **6414 kB** ($R = 571$)	354071 CC 12843 kB ($m_0 = 1721$, $m_1 = 7$)	$\times 0.998$ $\times 2.002$



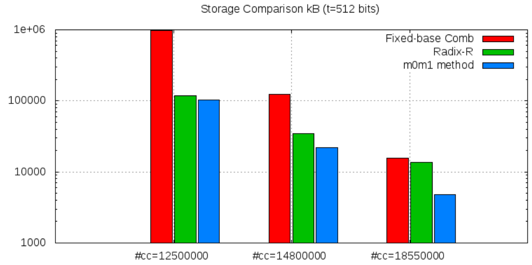
Performances

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 CC 1918 kB ($w = 11$)	4492191 CC 3430 kB ($R = 53$)	**4409584 CC** **1134 kB** ($m_0 = 23$, $m_1 = 10$)	$\times$0.993 $\times$0.591
3554339 CC 15358 kB ($w = 14$)	**3524896 CC** 8290 kB ($R = 163$)	3551437 CC **4164 kB** ($m_0 = 113$, $m_1 = 10$)	$\times$1.008 $\times$0.502
2736341 CC 245758 kB ($w = 18$)	**2543480 CC** 45221 kB ($R = 1223$)	2743399 CC **29961 kB** ($m_0 = 1031$, $m_1 = 7$)	$\times$1.079 $\times$0.662



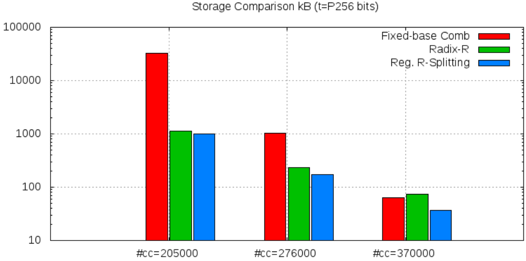
Performances

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



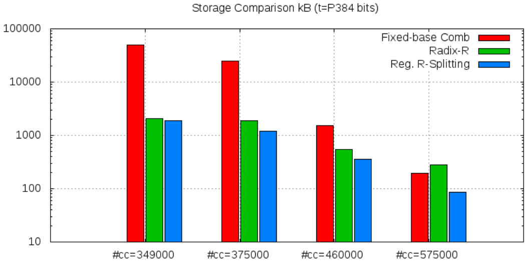
Performances

Level of Clock-cycles	Security level: 128 bits (NIST curve P256) Scalar multiplication — State of the art methods — Fixed-base Comb: Time (#CC)	Storage (kB)	w	radix R: Time (#CC)	Storage (kB)	R	Proposed approach R-splitting rec.: Time (#CC)	Storage (kB)	(R,c)
370000	378184	64	12	376370	74	19	366057	**37**	(71,5)
276000	275230	1024	14	276917	231	89	276660	**170**	(257,3)
205000	207456	32768	19	206777	1120	641	203414	**1012**	(1699,2)



Performances

	Security level: 192 bits (NIST curve P384) Scalar multiplication								
Level of Clock-cycles	State of the art methods						Proposed approach R-splitting rec.		
	Fixed-base Comb			radix R					
	Time (#CC)	Storage (kB)	w	Time (#CC)	Storage (kB)	R	Time (#CC)	Storage (kB)	(R, c)
575000	575854	192	11	571975	283	41	583590	**86**	(79,5)
460000	461271	1536	14	470537	547	97	451846	**354**	(233,3)
375000	376114	24576	18	372952	1861	433	378733	**1214**	(997,3)
349000	359578	49151	19	360786	2069	491	354919	**1911**	(1699,3)



Performances

Level of Clock-cycles	Fixed-base Comb Time (#CC)	Fixed-base Comb Storage (kB)	w	radix R Time (#CC)	radix R Storage (kB)	R	R-splitting rec. Time (#CC)	R-splitting rec. Storage (kB)	(R, c)
	Security level: 256 bits (NIST curve P521) — Scalar multiplication								
	State of the art methods						Proposed approach		
450000	446633	288	11	451280	572	41	449550	**146**	(97,7)
364000	363615	2304	14	362166	1621	157	367299	**726**	(433,5)
289000	289085	73728	19	288394	7217	937	290146	**6243**	(2897,3)

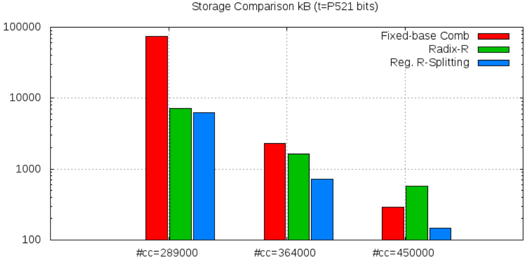


Table des matières

Conclusion

⟶ We have presented:

- Main State of the Art approaches for modular exponentiation;
- Our Contributions:
  - $m_0 m_1$ RNS digit recoding for exponent;
  - Enhanced algorithms for modular exponentiation;
  - R-splitting (alternative to the $m_0 m_1$ recoding);
  - Improvements to thwart side-channel analysis (timing attacks...);
  - Application to ECDSA (Elliptic Curve Digital Signature Algorithm);
  - Software implementations;
- This work has been accepted for publication in the JCEN.

Je vous remercie de votre attention,

et suis à l'écoute de vos questions ?