Efficient Modular Exponentiation Based on Multiple Multiplications by a Common Operand

Christophe NEGRE[(1)], Thomas PLANTARD[(2)]
and Jean-Marc ROBERT[(1)]

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

Arith22 2015, Lyon, the 22-24th of June 2015

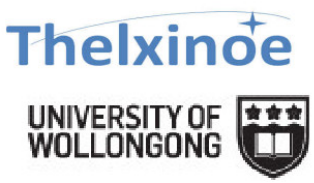


Table of Content

Table of Content

RSA Protocol (Rivest, Shamir and Adlemann):

Alice generates the keys. She:

- Chooses two distinct prime numbers p and q;
- Computes $N = pq$;
- Computes $\phi(N) = \phi(p)\phi(q) = (p-1)(q-1) = N - (p+q-1)$;
- Chooses an integer e such that $1 < e < \phi(N)$ and $gcd(e, \phi(N)) = 1$;
- Solves for d given $d \cdot e \equiv 1 \mod \phi(N)$;

$\rightarrow$ e is released as the public key exponent,

$\rightarrow$ d is kept as the private key exponent.

RSA Protocol (2)

$\rightarrow$ Bob encrypts using Alice's public key e:

$$C = M^e \mod N$$

RSA Protocol (2)

→ Bob encrypts using Alice's public key e:

$$C = M^e \mod N$$

→ Alice decrypts Bob's message using her secret key d:

$$\begin{aligned} C^d \mod N &= (M^e)^d \mod N \\ &= M^{e \cdot d} \mod N \\ &= M^{1 \mod \phi(N)} \mod N \\ &= M \end{aligned}$$

RSA Protocol (2)

→ Bob encrypts using Alice's public key e:

$$C = M^e \mod N$$

→ Alice decrypts Bob's message using her secret key d:

$$\begin{aligned} C^d \mod N &= (M^e)^d \mod N \\ &= M^{e \cdot d} \mod N \\ &= M^{1 \mod \phi(N)} \mod N \\ &= M \end{aligned}$$

→ The main operation is the Modular Exponentiation

Table of Content

Square-and-multiply

We consider an RSA modulus N such that $N < 2^{wn}$.

Square-and-multiply

Require: N the RSA modulus, g and $e = (e_{k-1}, \ldots, e_0)_2$ integers $\in [0, \ldots, N[$, with $e_{k-1} = 1$.
Ensure: $X = g^e \mod N$

Left-to-right

$X \leftarrow g$
for $i = k-2$ downto 0 **do**
  $X \leftarrow X^2 \mod N$
  if $e_i = 1$ **then**
    $X \leftarrow X \cdot g \mod N$
return $(X = g^e)$

Right-to-left

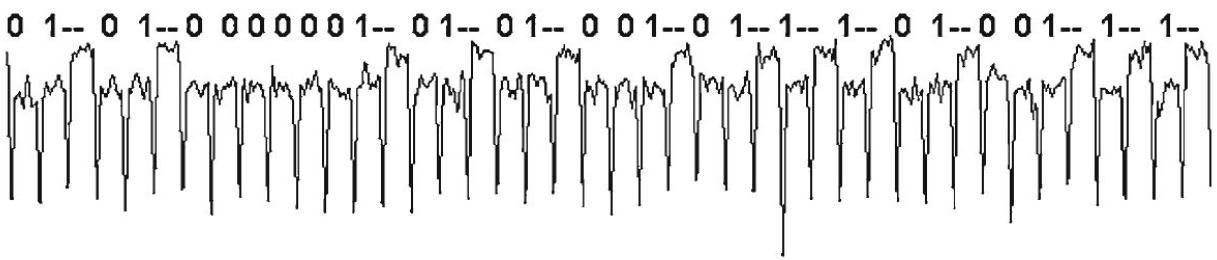
$X \leftarrow 1$
for $i = 0$ to $k-1$ **do**
  if $e_i = 1$ **then**
    $X \leftarrow X \cdot g \mod N$
  $g \leftarrow g^2 \mod N$
return $(X = g^e)$

Table of Content

Simple Power Analysis

RSA Left-to-right Square-and-multiply

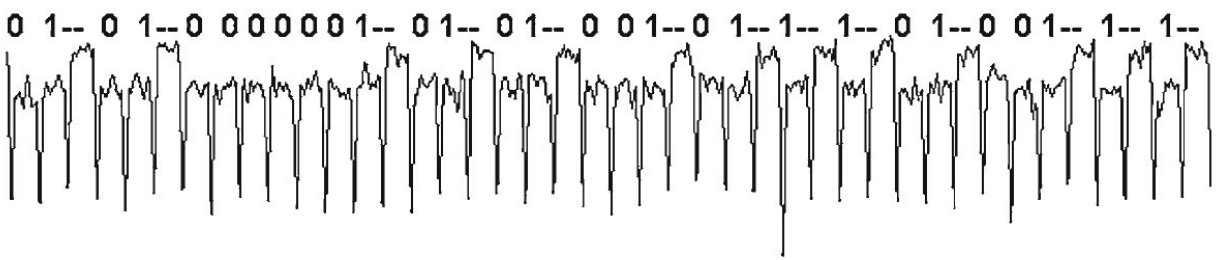
Require: N the RSA modulus, g and $e = (e_{k-1}, \ldots, e_0)_2$ integers $\in [0, \ldots, N[$, with $e_{k-1} = 1$.
Ensure: $X = g^e \mod N$
  $X \leftarrow g$
  for $i = k - 2$ downto 0 **do**
    $X \leftarrow X^2 \mod N$
    if $e_i = 1$ **then**
      $X \leftarrow X \cdot g \mod N$
  return $(X = g^e)$



Simple Power Analysis

RSA Left-to-right Square-and-multiply

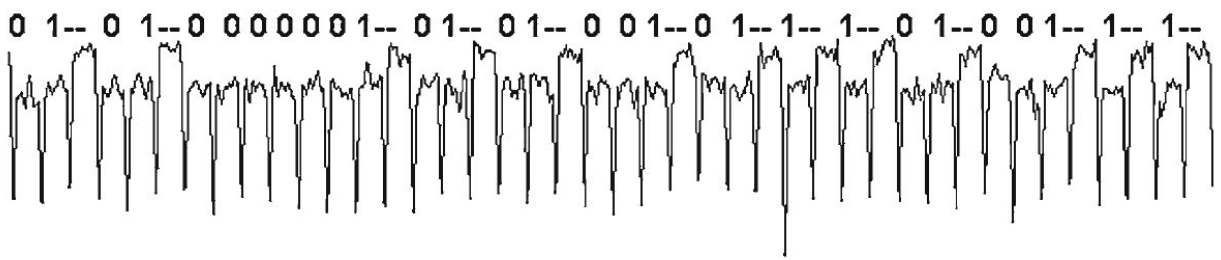
Require: N the RSA modulus, g and $e = (e_{k-1}, \ldots, e_0)_2$ integers $\in [0, \ldots, N[$, with $e_{k-1} = 1$.
Ensure: $X = g^e \mod N$
  $X \leftarrow g$
  for $i = k - 2$ downto 0 **do**
    $X \leftarrow X^2 \mod N$ $\rightarrow$ A squaring corresponds to a low crenel
    if $e_i = 1$ **then**
      $X \leftarrow X \cdot g \mod N$
  return $(X = g^e)$



Simple Power Analysis

RSA Left-to-right Square-and-multiply

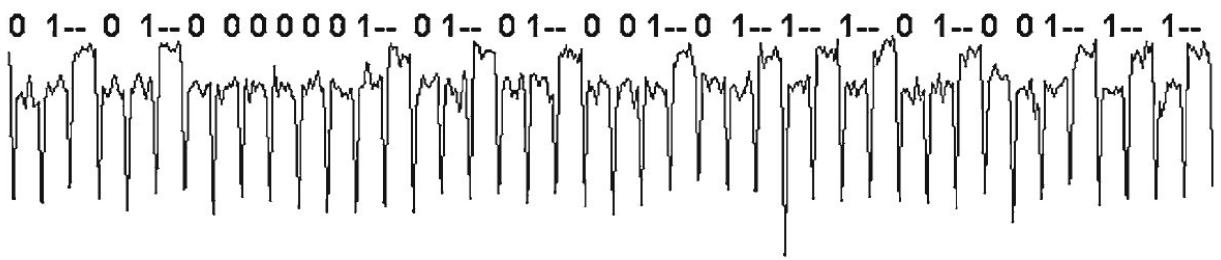
Require: N the RSA modulus, g and $e = (e_{k-1}, \ldots, e_0)_2$ integers $\in [0, \ldots, N[$, with $e_{k-1} = 1$.
Ensure: $X = g^e \mod N$
  $X \leftarrow g$
  for $i = k - 2$ downto 0 **do**
    $X \leftarrow X^2 \mod N$
    if $e_i = 1$ **then**
      $X \leftarrow X \cdot g \mod N$ $\rightarrow$ A multiplication corresponds to a high crenel
  return $(X = g^e)$



Simple Power Analysis

RSA Left-to-right Square-and-multiply

Require: N the RSA modulus, g and $e = (e_{k-1}, \ldots, e_0)_2$ integers $\in [0, \ldots, N[$, with $e_{k-1} = 1$.
Ensure: $X = g^e \mod N$
  $X \leftarrow g$
  for $i = k - 2$ downto 0 **do**
    $X \leftarrow X^2 \mod N$
    if $e_i = 1$ **then**
      $X \leftarrow X \cdot g \mod N$
  return $(X = g^e)$



$\rightarrow$ Vulnerable: the sequence of operations leaks the secret scalar (no regularity.)

Montgomery Binary Ladder

Montgomery

Require: $e = (e_{t-1}, \ldots, e_1, e_0)$ with $e_{t-1} = 1, g \in \mathbb{Z}/N\mathbb{Z}$
Ensure: $X = g^e \mod N$

1: $X_0 \leftarrow g, X_1 \leftarrow g^2 \mod N$
2: **for** i from $t-2$ downto 0 **do**
3: **if** $(e_i = 0)$ **then**
4: $X_1 \leftarrow X_0 \cdot X_1 \mod N, X_0 \leftarrow X_0^2 \mod N$
5: **else**
6: $X_0 \leftarrow X_0 \cdot X_1 \mod N, X_1 \leftarrow X_1^2 \mod N$
7: **return** (X_0)

Basic Montgomery's Ladder Modular Exponentiation

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Then, the addition without carry leads to the 0-less recoding:

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3

Then, the addition without carry leads to the 0-less recoding:

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1

Then, the addition without carry leads to the 0-less recoding:

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$
1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Joye and Tunstall suggested a 2^t-ary recoding without zero digits:

Unsigned-Digit Recoding Algorithm

Require: $e \geq 1$, $m = 2^t$, ℓ the m-ary length of e and N the RSA modulus
Ensure: $e = (e_{\ell-1}, \ldots, e_0)$ with $e_i \in \{1, \ldots, m\}$, $1 \leq i \leq \ell - 2$

1: $s \leftarrow (1, 1, \ldots, 1)_m$
2: $e' \leftarrow e - s \mod N$
3: **for** $i = 0$ to $\ell - 2$ **do**
4: $\quad d \leftarrow e' \mod m$
5: $\quad e' \leftarrow \lfloor e'/m \rfloor$
6: $\quad e_i \leftarrow d + 1$

Example with $t = 4, m = 10_{10}$:

e	$=$	1	9	7	5	0	4	0	2	3
s	$=$	1	1	1	1	1	1	1	1	1
$e' \leftarrow e - s \mod N$		0	8	6	3	9	2	9	1	2

Then, the addition without carry leads to the 0-less recoding:

$e' \mod N$		0	8	6	3	9	2	9	1	2
s	$=$	1	1	1	1	1	1	1	1	1
recoding		1	9	7	4	10	3	10	2	3

Regular Exponentiation Algorithms

Regular 2^t-ary Square-and-multiply

Require: g, Joye-Tunstall recoding of $e = (e_{k-1}, \ldots, e_0) < N$
Ensure: $X = g^e \mod N$

Left-to-right

$Y_1 \leftarrow g$
for $i = 2$ to 2^t **do**
  $Y_i \leftarrow Y_{i-1} \cdot g \mod N$
$X \leftarrow Y_{e_{k-1}}$
for $i = k-2$ downto 0 **do**
  $X \leftarrow X^{2^t} \mod N$
  $X \leftarrow X \cdot Y_{e_i} \mod N$
return $(X = g^e)$

Right-to-left

$X \leftarrow g, Y_i \leftarrow 1, i \in \{1, \ldots, 2^t\}$
for $i = 0$ to $k-1$ **do**
  $Y_{e_i} \leftarrow Y_{e_i} \cdot X \mod N$
  $X \leftarrow X^{2^t} \mod N$
// Final reconstruction
$X \leftarrow Y_{2^t}$
for $i = 2^t - 1$ downto 1 **do**
  $Y_i \leftarrow Y_i \cdot Y_{i+1} \mod N$
  $X \leftarrow X \cdot Y_i \mod N$
return $(X = g^e)$

Table of Content

Montgomery Modular Multiplication

Let A and B in $[0, N[$. Let us set $C = A \times B, C < N^2$.
Montgomery reduction of $C = A \times B$:

Montgomery Modular Multiplication

Let A and B in $[0, N[$. Let us set $C = A \times B, C < N^2$.
Montgomery reduction of $C = A \times B$:

One set $Q \leftarrow A \times B \times (-N^{-1}) \mod 2^{wn}$, then the division

$$Y \leftarrow (A \times B + Q \times N)/2^{wn}.$$

is exact.

Montgomery Modular Multiplication

Let A and B in $[0, N[$. Let us set $C = A \times B, C < N^2$.
Montgomery reduction of $C = A \times B$:

One set $Q \leftarrow A \times B \times (-N^{-1}) \mod 2^{wn}$, then the division

$$Y \leftarrow (A \times B + Q \times N)/2^{wn}.$$

is exact.

Y satisfies

$$Y = A \times B \times 2^{-wn} \mod N \text{ and } Y < 2N.$$

Montgomery Modular Multiplication

Let A and B in $[0, N[$. Let us set $C = A \times B, C < N^2$.
Montgomery reduction of $C = A \times B$:

One set $Q \leftarrow A \times B \times (-N^{-1}) \mod 2^{wn}$, then the division

$$Y \leftarrow (A \times B + Q \times N)/2^{wn}.$$

is exact.

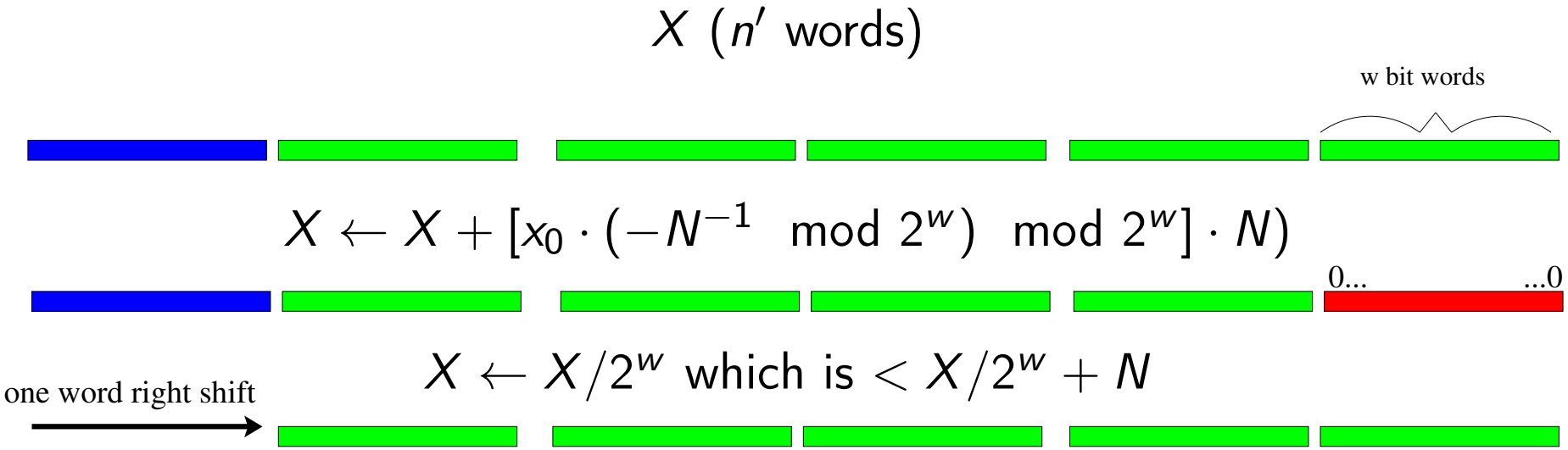
Y satisfies

$$Y = A \times B \times 2^{-wn} \mod N \text{ and } Y < 2N.$$

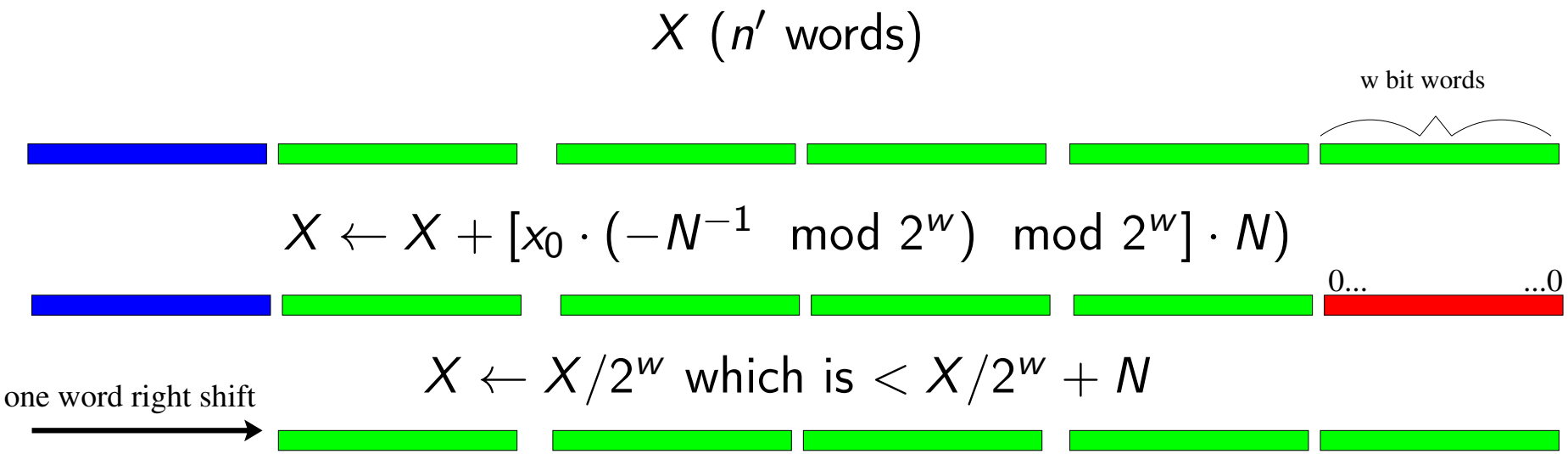
Montgomery representation: $\widetilde{A} = A \times 2^{wn} \mod N$.

$$\widetilde{A} \times \widetilde{B} \times 2^{-wn} \mod N = (AB) \times 2^{wn} \mod N$$

SmallRed: One Word Reduction



SmallRed: One Word Reduction



SmallRed (one word)

Require: A modulus $N < 2^{wn-2}$ and a positive integer $X = (x_{n'-1}, \ldots, x_0)_{2^w}$ of n' words and $N' = (-N^{-1}) \mod 2^w$

Ensure: $Y = X \cdot 2^{-w} \mod N$ with $Y < X/2^w + N$

1: $q \leftarrow x_0 \cdot N' \mod 2^w$
2: $Y \leftarrow (X + q \cdot N)/2^w$
3: **return** Y

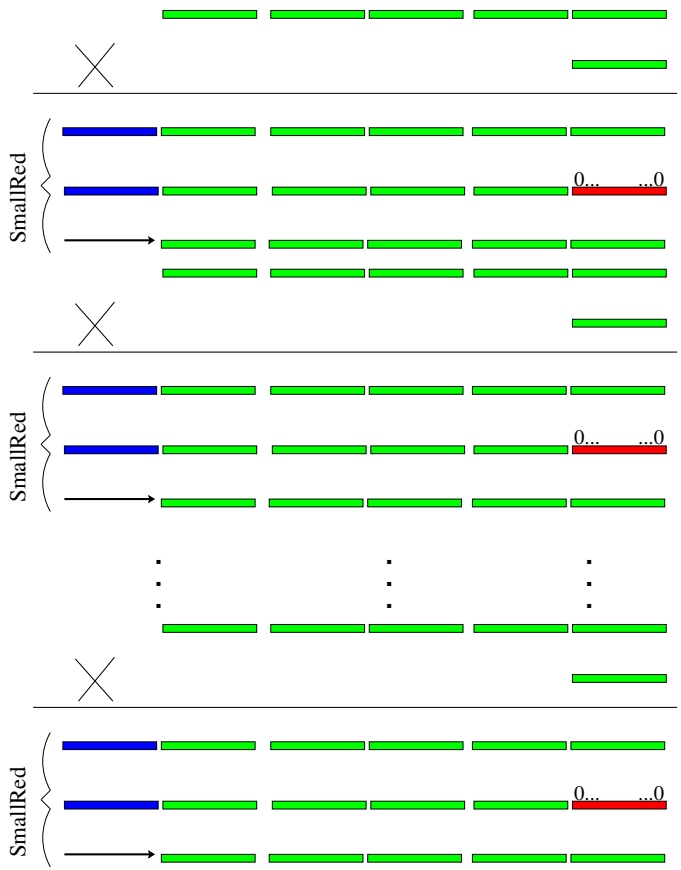
Block Montgomery Multiplication

$$ABR^{-1} \mod N = \left(\sum_{j=0}^{n-1} b_j 2^{wj}\right) \cdot A \cdot 2^{-wn} \mod N$$

Block Montgomery Multiplication

$$ABR^{-1} \mod N = \left(\sum_{j=0}^{n-1} b_j 2^{wj}\right) \cdot A \cdot 2^{-wn} \mod N$$

$$\begin{cases} A \cdot b_0 \cdot 2^{-w} \\ (A \cdot b_0 \cdot 2^{-w} + A \cdot b_1) \cdot 2^{-w} \\ \left(\sum_{j=0}^{n-1} b_j 2^{wj}\right) \cdot A \cdot 2^{-wn} \end{cases}$$



Block Montgomery Multiplication

Complexity:

operation	# word-multiplications	# word additions
$ABR^{-1} \mod N$	$2n^2 + n$	$4n^2 + 2n - 1$
$A^2R^{-1} \mod N$	$\frac{3}{2}n^2 + \frac{5}{2}n - 1$	$3n^2 + 5n - 1$

Table of Content

Our Objective

Our objective is to speed-up the SPA-resistant algorithms.

To achieve this goal we focused on:

Our Objective

Our objective is to speed-up the SPA-resistant algorithms.

To achieve this goal we focused on:

- Two Modular Multiplications by a common operand $A \cdot B, A \cdot C$;

Our Objective

Our objective is to speed-up the SPA-resistant algorithms.

To achieve this goal we focused on:

- Two Modular Multiplications by a common operand $A \cdot B, A \cdot C$;
- Multiple Modular Multiplications by a common operand $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$;

Our Objective

Our objective is to speed-up the SPA-resistant algorithms.

To achieve this goal we focused on:

- Two Modular Multiplications by a common operand $A \cdot B, A \cdot C$;
- Multiple Modular Multiplications by a common operand $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$;
- Application to SPA protected modular exponentiations.

Table of Content

CombinedMontMul $A \cdot B, A \cdot C$

The Block Montgomery Modular multiplication computes:

$$\begin{aligned} A \cdot B \cdot R^{-1} &= (\sum_{j=0}^{n-1} b_j 2^{wj}) \cdot A \cdot 2^{-w(n+1)} \mod N \\ &= \sum_{j=0}^{n-1} b_j A \cdot 2^{-w(n+1-j)} \mod N \\ &= \sum_{j=0}^{n-1} b_j (A \cdot 2^{-w(n-1-j)} \mod N) 2^{-2w} \mod N \\ &= (\sum_{j=0}^{n-1} b_j A^{(j)}) \cdot 2^{-2w} \mod N \end{aligned} \quad (1)$$

where $A^{(j)} = A2^{-w(n-1-j)} \mod N$ for $j = 0, \ldots, n-1$. With the same for $A \cdot C \cdot R^{-1}$, one obtains

$$A \cdot C \cdot R^{-1} = (\sum_{j=0}^{n-1} c_j A^{(j)}) \cdot 2^{-2w} \mod N. \quad (2)$$

CombinedMontMul $A \cdot B, A \cdot C$

The Block Montgomery Modular multiplication computes:

$$\begin{aligned} A \cdot B \cdot R^{-1} &= (\sum_{j=0}^{n-1} b_j 2^{wj}) \cdot A \cdot 2^{-w(n+1)} \mod N \\ &= \sum_{j=0}^{n-1} b_j A \cdot 2^{-w(n+1-j)} \mod N \\ &= \sum_{j=0}^{n-1} b_j (A \cdot 2^{-w(n-1-j)} \mod N) 2^{-2w} \mod N \\ &= (\sum_{j=0}^{n-1} b_j A^{(j)}) \cdot 2^{-2w} \mod N \end{aligned} \quad (1)$$

where $A^{(j)} = A2^{-w(n-1-j)} \mod N$ for $j = 0, \ldots, n-1$. With the same for $A \cdot C \cdot R^{-1}$, one obtains

$$A \cdot C \cdot R^{-1} = (\sum_{j=0}^{n-1} c_j A^{(j)}) \cdot 2^{-2w} \mod N. \quad (2)$$

CombinedMontMul $A \cdot B, A \cdot C$: Shared Computations

Therefore, the values $A^{(j)}$ are shared by both multiplications:

$$\begin{aligned}
A &= A^{(n-1)} \\
\mathrm{SmallRed}(A^{(n-1)}) &= A \cdot 2^{-w} \mod N \\
&= A^{(n-2)}, \\
\mathrm{SmallRed}(A^{(n-2)}) &= (A \cdot 2^{-w}) \cdot 2^{-w} \mod N \\
&= A^{(n-3)}, \\
&\vdots \quad\quad \vdots \\
\mathrm{SmallRed}(A^{(1)}) &= (A \cdot 2^{-w(n-2)}) \cdot 2^{-w} \mod N \\
&= A \cdot 2^{-w(n-1)} = A^{(0)}.
\end{aligned}$$

CombinedMontMul $A \cdot B, A \cdot C$: Algorithm

CombinedMontMul(A, B, C)

Require: the modulus $N < 2^{wn-2}$, three integers $A = (a_{n-1}, \ldots a_0)_2, B = (b_{n-1}, \ldots b_0)_2, C = (c_{n-1}, \ldots c_0)_2$ such that $A, B, C < 2N$, w the word size, $R = 2^{w(n+1)}$ the Montgomery constant.

Ensure: $Y = A \cdot B \cdot R^{-1} \mod N$ and $Z = A \cdot C \cdot R^{-1} \mod N$

1: $X \leftarrow A$
2: $Y \leftarrow b_{n-1} \cdot X,\ Z \leftarrow c_{n-1} \cdot X$
3: **for** $j = n - 2$ **downto** 0 **do**
4: $\quad q \leftarrow |X|_{2^w} N' \mod 2^w$
5: $\quad X \leftarrow (X + q \cdot N)/2^w \quad // = A^{(j)}$ for $j = n-2, \ldots, 0$
6: $\quad Y \leftarrow Y + b_j \cdot X,\ Z \leftarrow Z + c_j \cdot X$
7: $Y \leftarrow \text{SmallRed}(Y),\ Z \leftarrow \text{SmallRed}(Z)$
8: $Y \leftarrow \text{SmallRed}(Y),\ Z \leftarrow \text{SmallRed}(Z)$
9: **return** Y and Z

CombinedMontMul $A \cdot B, A \cdot C$: Complexity Comparison

By sharing the computations of $A^{(j)}$, we reduce the complexity of the global computation of both multiplications $A \cdot B, A \cdot C$:

Operation	Algorithm	# ADD	# MUL
AB, AC	Two MontMuls	$8n^2 + 4n - 2$	$4n^2 + 2n$
AB, A^2	MontMul and MontSqu	$7n^2 + 7n - 2$	$\frac{7}{2}n^2 + \frac{7}{2}n - 1$
AB, AC	CombinedMontMul	$6n^2 + 9n + 1$	$3n^2 + 4n + 3$

Table of Content

$A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$

Given $A \in \{0, \ldots, N-1\}$ and $B_i, i = 1, \ldots, \ell$, we want to compute:

$$\begin{aligned} Y_1 &= A \cdot B_1 \cdot 2^{-w(n+1)} \mod N, \\ Y_2 &= A \cdot B_2 \cdot 2^{-w(n+1)} \mod N, \\ &\vdots \\ Y_\ell &= A \cdot B_\ell \cdot 2^{-w(n+1)} \mod N. \end{aligned}$$

$A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$

Given $A \in \{0, \ldots, N-1\}$ and $B_i, i = 1, \ldots, \ell$, we want to compute:

$$\begin{aligned} Y_1 &= A \cdot B_1 \cdot 2^{-w(n+1)} \mod N, \\ Y_2 &= A \cdot B_2 \cdot 2^{-w(n+1)} \mod N, \\ &\vdots \\ Y_\ell &= A \cdot B_\ell \cdot 2^{-w(n+1)} \mod N. \end{aligned}$$

We expand each multiplication $A \cdot B_i \cdot 2^{-w(n+1)} \mod N$ relatively to B_i and rewrite the product as follows:

$$\begin{aligned} A \cdot B_i \cdot 2^{-w(n+1)} &= \left(\sum_{j=0}^{n-1} b_{i,j} 2^{wj}\right) \cdot A^{-w(n+1)} \mod N \\ &= \left(\sum_{j=0}^{n-1} b_{i,j} A^{(j)}\right) \cdot 2^{-2w} \mod N. \end{aligned} \tag{3}$$

Optimisations $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$: Complexity

Operation	Algorithm	# ADD	# MUL
AB_i, $i = 1, \ldots, \ell$	ℓ MontMuls	$\ell(4n^2 + 2n - 1)$	$\ell(2n^2 + n)$
AB_i, $i = 1, \ldots, \ell$	$\ell\times$ MultByComOp and $1\times$ PrecomMultByComOp	$\ell(2n^2 + 5n + 1)$ $+(2n + 1)(n - 1)$	$\ell(n^2 + 2n + 2)$ $+(n^2 - 1)$

Optimisations $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$: Complexity

Operation	Algorithm	# ADD	# MUL
AB_i, $i = 1, \ldots, \ell$	ℓ MontMuls	$\ell(4n^2 + 2n - 1)$	$\ell(2n^2 + n)$
AB_i, $i = 1, \ldots, \ell$	$\ell\times$ MultByComOp and $1\times$ PrecomMultByComOp	$\ell(2n^2 + 5n + 1)$ $+(2n+1)(n-1)$	$\ell(n^2 + 2n + 2)$ $+(n^2 - 1)$

Optimisations $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$: Complexity

Operation	Algorithm	# ADD	# MUL
AB_i, $i = 1, \ldots, \ell$	ℓ MontMuls	$\ell(4n^2 + 2n - 1)$	$\ell(2n^2 + n)$
AB_i, $i = 1, \ldots, \ell$	$\ell\times$ MultByComOp and $1\times$ PrecomMultByComOp	$\ell(2n^2 + 5n + 1)$ $+(2n+1)(n-1)$	$\ell(n^2 + 2n + 2)$ $+(n^2 - 1)$

Optimisations $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$: Complexity

Operation	Algorithm	# ADD	# MUL
AB_i, $i = 1, \ldots, \ell$	ℓ MontMuls	$\ell(4n^2 + 2n - 1)$	$\ell(2n^2 + n)$
AB_i, $i = 1, \ldots, \ell$	$\ell\times$ MultByComOp and $1\times$ PrecomMultByComOp	$\ell(2n^2 + 5n + 1)$ $+(2n+1)(n-1)$	$\ell(n^2 + 2n + 2)$ $+(n^2 - 1)$

PrecomMultByComOp $\rightarrow$ computations of all $A^{(j)}$.

Storage required: $\rightarrow$ $n \times (n$ word operand $A^{(j)})$.

Table of Content

Montgomery Ladder with $A \cdot B, A \cdot C$

Montgomery Ladder with CombinedMontMul(A, B, C)

Require: $N < 2^{wn-1}$ and $G \in \{0, \ldots, N-1\}$ and an exponent $e = (e_{k-1}, \ldots e_0)_2$, w the word size and $R = 2^{w(n+1)}$ the Montgomery constant.

Ensure: $G^e \mod N$

1: $X_0 \leftarrow R \mod N$
2: //conversion $X_1 \leftarrow G \cdot R^2 \cdot R^{-1} \mod N$
3: $X_1 \leftarrow \text{MontMul}(G, R^2 \mod N)$
4: $X_1 \leftarrow \text{SmallRed}(X_1)$
5: **for** $i = k-1$ **downto** 0 **do**
6: $\quad X_{1-e_i}, X_{e_i} \leftarrow \text{CombinedMontMul}(X_{e_i}, X_{1-e_i}, X_{e_i})$
7: // conversion $X_0 \leftarrow (G^e \cdot R) \cdot R^{-1} \mod N$
8: $X_0 \leftarrow \text{MontMul}(X_0, 1)$, $X_0 \leftarrow \text{SmallRed}(X_0)$
9: **return** X_0

Right-to-left Regular Exponentiation with $A \cdot B, A \cdot C$

Right-to-left regular 2^t-ary exponentiation with CombinedMontMul

Require: $N < 2^{wn-2}$ the modulus, an integer $0 \leq G < N$, an exponent $e = (e_{k-1}, \ldots, e_0)_{2^t}$ with $e_i \in \{1, \ldots, 2^t\}$, $R = 2^{w(n+1)}$ the Montgomery constant.
Ensure: $G^e \mod N$

1: $X \leftarrow \text{MontMul}(G, R^2 \mod N)$, $X \leftarrow \text{SmallRed}(X)$ // $X = G \cdot R \mod N$
2: **for** $i = 1$ to 2^t **do**
3: $\quad Y_i \leftarrow R \mod N$
4: **for** $i = 0$ to $k - 1$ **do**
5: $\quad Y_{e_i}, X \leftarrow \text{CombinedMontMul}(X, Y_{e_i}, X)$
6: $\quad$ **for** $j = 1$ **to** $t - 1$ **do**
7: $\quad\quad X \leftarrow \text{MontSqu}(X)$, $X \leftarrow \text{SmallRed}(X)$
8: // Final reconstruction
9: $Z \leftarrow Y_{2^t}$
10: $Y_{2^t-1} \leftarrow \text{MontMul}(Y_{2^t-1}, Y_{2^t})$
11: $Y_{2^t-1} \leftarrow \text{SmallRed}(Y_{2^t-1})$
12: **for** $i = 2^t - 1$ **downto** 2 **do**
13: $\quad Z, Y_{i-1} \leftarrow \text{CombinedMontMul}(Y_i, Z, Y_{i-1})$
14: $Z \leftarrow \text{MontMul}(Z, Y_1)$, $Z \leftarrow \text{SmallRed}(Z)$
15: $Z \leftarrow \text{MontMul}(Z, 1)$, $Z \leftarrow \text{SmallRed}(Z)$
16: **return** Z

Left-to-right Regular Exponentiation with $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$

Left-to-right regular 2^t-ary exponentiation with MultByComOp

Require: $N < 2^{wn-1}$ the modulus, an integer $0 \leq G < N$, an exponent $e = (e_{k-1}, \ldots, e_0)_2$ with $e_{k-1} \in \{1, \ldots, m\}$, $R = 2^{w(n+1)}$ the Montgomery constant.
Ensure: $G^e \mod N$

1: $G_1 \leftarrow \text{MontMul}(G, R^2 \mod N), G_1 \leftarrow \text{SmallRed}(G_1)$// $G_1 = G \cdot 2^{w(n+1)} \mod N$
2: $X \leftarrow R$
3: $G_1^{(0)}, \ldots, G_1^{(n-1)} \leftarrow \text{PrecompMultByComOp}(G_1)$
4: **for** $i = 2$ **to** 2^t **do**
5: $\quad G_i \leftarrow \text{MultByComOp}(G_{i-1}, G_1^{(0)}, \ldots, G_1^{(n-1)})$
6: $\quad G_i^{(0)}, \ldots, G_i^{(n-1)} \leftarrow \text{PrecompMultByComOp}(G_i)$
7: **for** $i = k - 1$ **downto** 0 **do**
8: $\quad$ **for** $j = 1$ **to** t **do**
9: $\quad\quad X \leftarrow \text{MontSqu}(X), X \leftarrow \text{SmallRed}(X)$// $X \leftarrow X^2 \cdot 2^{-w(n+1)} \mod N$
10: $\quad X \leftarrow \text{MultByComOp}(X, G_{e_i}^{(0)}, \ldots, G_{e_i}^{(n-1)})$
11: $X \leftarrow \text{MontMul}(X, 1), X \leftarrow \text{SmallRed}(X)$// $X \leftarrow X \cdot 2^{-w(n+1)} \mod N$
12: **return** X

Complexity Comparison

Complexities of modular exponentiation for 2048 bits

	ML	ML CMM	R-to-L	R-to-L CMM	L-to-R	L-to-R MBCO
#ADD/10^3	15143	13183	8595	8253	8466	7804
Improv.	12.9%		4%		7.9%	
#MUL/10^3	7506	6564	4296	4120	4232	3896
Improv.	12.6%		4.1%		7.9%	
Storage req.	-		-		256 KB with $t = 5$	

ML=Montgomery-ladder, R-to-L= Right-to-left, L-to-R=Left-to-right

Table of Content

Implementation Performances

- Intel Core i7 (turbo-boost and hyperthreading deactivated), gcc 4.8.2
- `GMP 6.0.0` library for $1 \times n$ multiplications and $n \times n$ additions

Algorithm	1024 bits		2048 bits		4096 bits	
	#CC/10^3	Imp. ratio	#CC/10^3	Imp. ratio	#CC/10^3	Imp. ratio
Mont-ladder	3068	9.0%	20643	9.1%	153443	14.6%
Mont-ladder with CMM	2793		18773		131011	
Right-to-left	1857	0.1%	11796	1.7%	87081	4.0%
Right-to-left with CMM	1855		11596		83599	
Left-to-right	1858	-1%	11734	3.1 %	83354	8.3 %
Left-to-right with MBCO	1877		11368		77745	

Algorithm and Implementation Optimisation

Conclusion:

In this work, we proposed:

- Two Modular Multiplications by a common operand $A \cdot B, A \cdot C$:
  $\rightarrow$ Improvement of two modular multiplications up to 25%;

Conclusion:

In this work, we proposed:

- Two Modular Multiplications by a common operand $A \cdot B, A \cdot C$:
  $\rightarrow$ Improvement of two modular multiplications up to 25%;
- Multiple Modular Multiplications by a common operand $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$:
  $\rightarrow$ Improvement of ℓ modular multiplications up to 50%;

Conclusion:

In this work, we proposed:

- Two Modular Multiplications by a common operand $A \cdot B, A \cdot C$:
  $\rightarrow$ Improvement of two modular multiplications up to 25%;
- Multiple Modular Multiplications by a common operand $A \cdot B_0, A \cdot B_1, \ldots, A \cdot B_\ell$:
  $\rightarrow$ Improvement of ℓ modular multiplications up to 50%;
- Application to SPA protected RSA modular exponentiations:
  $\rightarrow$ the Montgomery Ladder is improved by 9 to 15 %;
  $\rightarrow$ the Right-to-left regular 2^t-ary exponentiation is improved by up to 4 %;
  $\rightarrow$ the Left-to-right regular 2^t-ary exponentiation is improved by up to 8 %.

Thank you for your attention,

Any questions ?

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

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

This work was supported by:

- PAVOIS ANR 12 BS02 002 02
- ERASMUS MUNDUS Thelxinoe Smart City