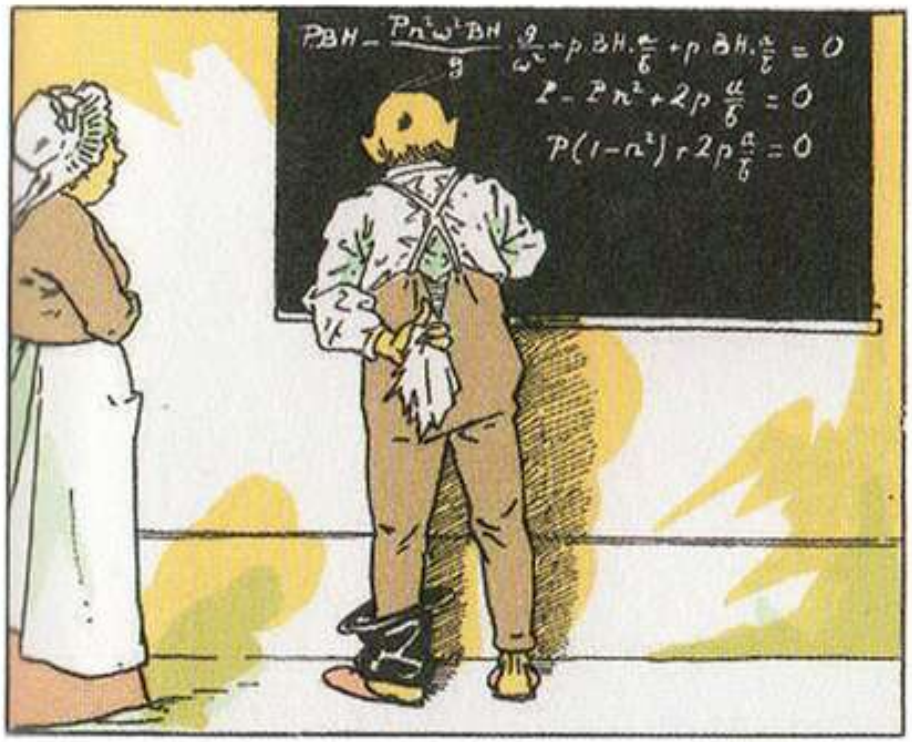
# Montgomery Truncated Multiplications and Batch Exponentiations using IFMA52 instructions

Laurent-Stéphane Didier, Nadia El Mrabet, Léa Glandus et **Jean-Marc Robert**

Octobre 2024

# L'idée de Scholastique (Christophe, 1899)

Complètement remis, Cosinus songe à reprendre ses études de cyclisme théorique et pratique, si fâcheusement interrompues par l'imprudence de Sphéroïde, vil instrument dont s'est servi la fatalité qui s'obstine à empêcher Cosinus de partir.



À cet effet, il se remet à étudier l'équilibre des corps en mouvement. Scholastique n'arrive pas à comprendre l'utilité qu'il peut y avoir à écrire des tas de choses pour arriver à mettre dans le bout : $= 0$. Autant vaudrait, à son avis, ne pas les écrire. Mais, en matière de sciences, l'opinion de Scholastique est négligeable.

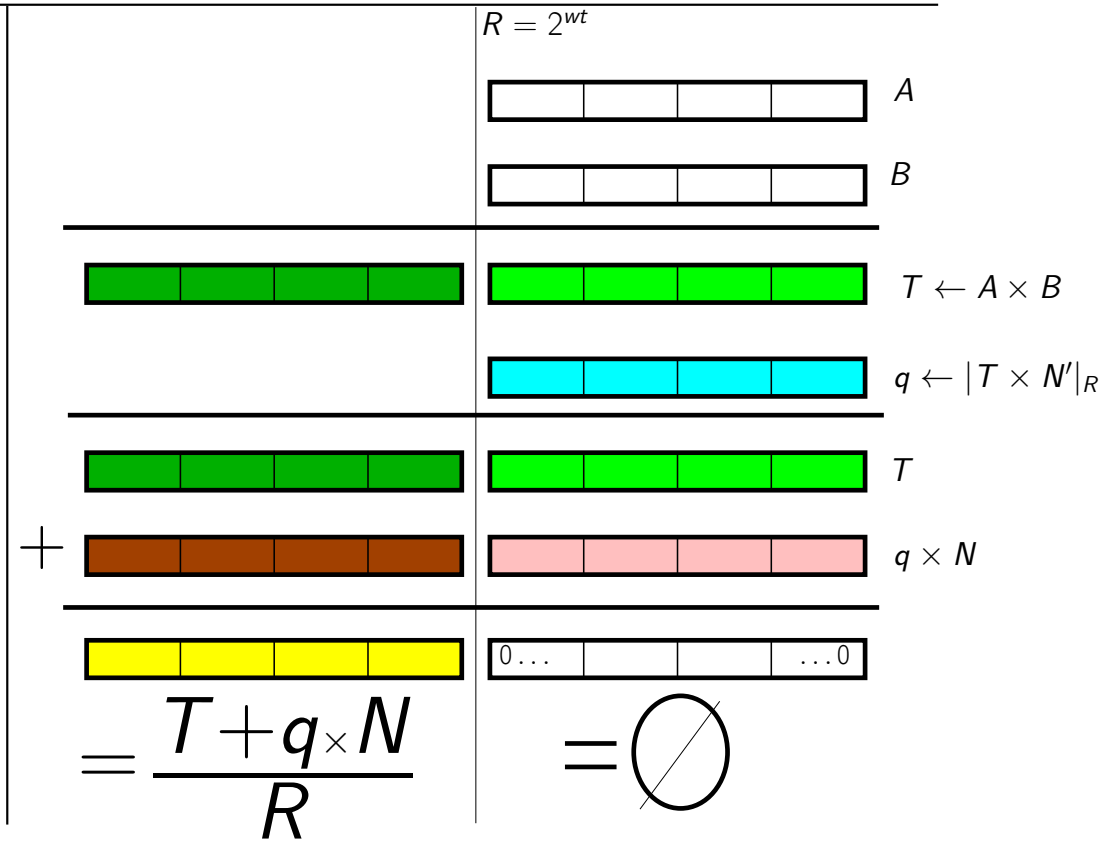
# Montgomery Multiplication : algorithm

**Require:** $T < 4N^2$, $N$ the $wt$-bit modulus, $R = 2^{wt}$,
precomputed value $N' = (-N)^{-1} \bmod R$.
**Ensure:** $C \equiv T \times R^{-1} \bmod N$ and $C < 2N$

$q \leftarrow T \times N' \bmod R$

$C \leftarrow \frac{T+q\times N}{R}$

**return** $C$



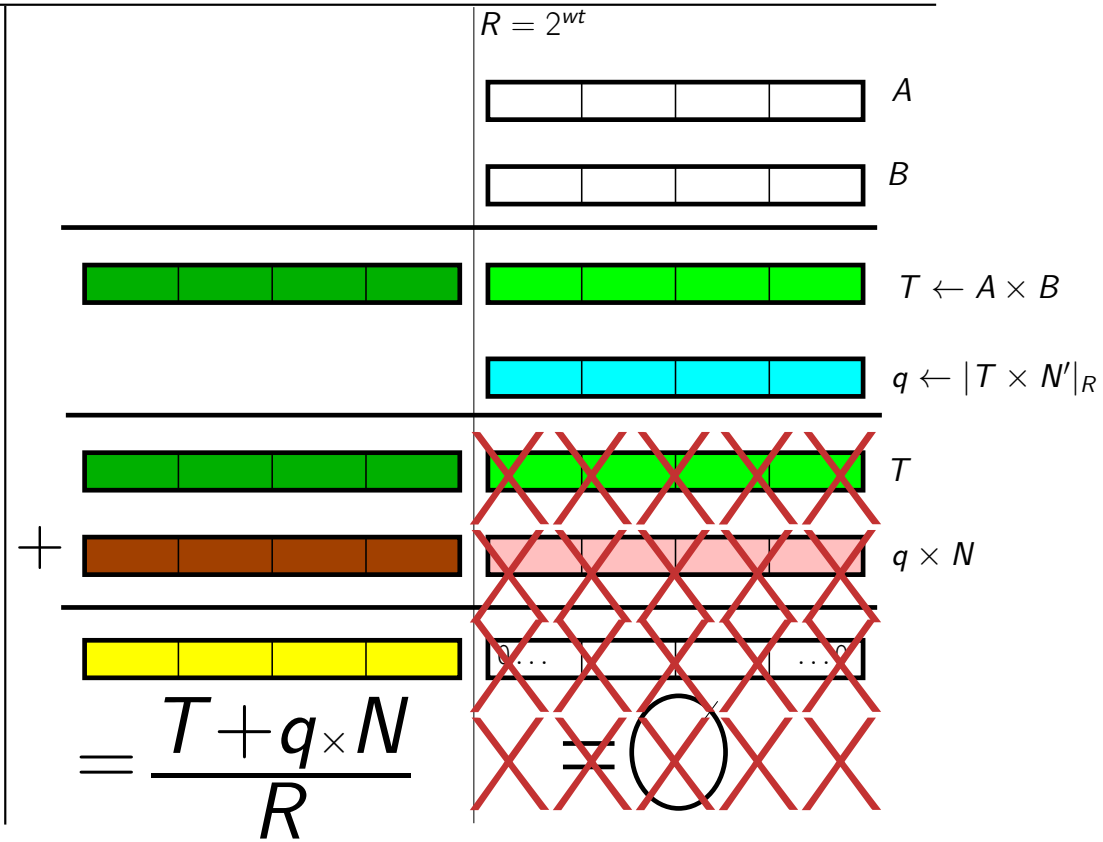
# Montgomery Multiplication : algorithm

**Require:** $T < 4N^2$, $N$ the $wt$-bit modulus, $R = 2^{wt}$,
precomputed value $N' = (-N)^{-1} \bmod R$.
**Ensure:** $C \equiv T \times R^{-1} \bmod N$ and $C < 2N$

$q \leftarrow T \times N' \bmod R$

$C \leftarrow \frac{T+q\times N}{R}$

**return** $C$



# History

Montgomery Multiplication:

- No mention of the idea (Montgomery 1985)
- Idea of a truncated operation mentioned (Barrett 1986)
- Block or word approaches in Montgomery modular reduction or multiplication (Koc et Al. 1996)
  - $\rightarrow$ Coarsely Integrated Operand Scanning CIOS
  - $\rightarrow$ Not suitable for truncated operations

# CIOS Montgomery multiplication

**Require:** Two values $A$ and $B$ , the modulus $N$, the precomputed value $N' = (-N)^{-1} \bmod R$ with $R = 2^t$, $w$ the word size.

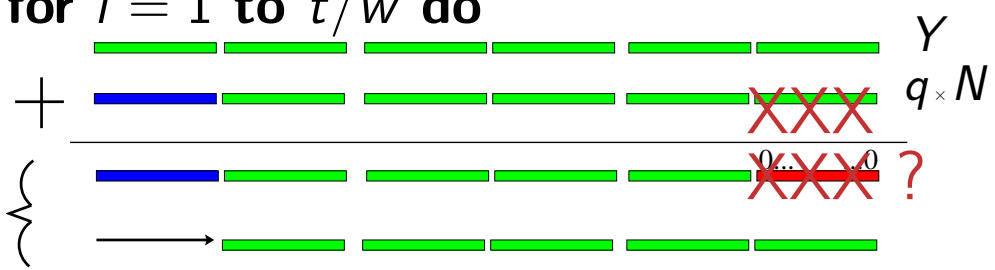
**Ensure:** $C = A \times B \times R^{-1} \bmod N$.

$Y \leftarrow a[0] \cdot B$
$q \leftarrow Y \cdot N' \mod 2^w$
$Y \leftarrow (Y + q \cdot N)/2^w$
**for** $i = 1$ **to** $t/w$ **do**
  $Y \leftarrow Y + a[i] \cdot B$
  $q \leftarrow Y \cdot N' \mod 2^w$
  $Y \leftarrow (Y + q \cdot N)/2^w$

**end for**

**return** $C \leftarrow Y$

**for** $i = 1$ **to** $t/w$ **do**




**end for**

# History (part 2)

Vectorized implementations of Montgomery CIOS:

- Vectorized implementations of Montgomery Multiplications using word-slicing representation (Bos *et al.* 2021) $\rightarrow$ No implementation results
- Batch implementations of Montgomery Multiplications using Montgomery friendly for SIKE (Cheng *et al.* 2022) $\rightarrow$ No generalization

Hardware Truncated Montgomery Multiplication:

- Theoretical work on truncated multiplications (Hars 2005-2006)
- Implementation of a 256 bit ECC processor using a Barrett modular truncated multiplication (Ding *et al.* 2018)
- FPGA implementations based on 3 and 4 way Karatsuba truncated multipliers (Ding *et al.* 2020)
  $\rightarrow$ Not easily transferable to software
  $\rightarrow$ Few versions and sizes $\Longrightarrow$ difficult to generalize.

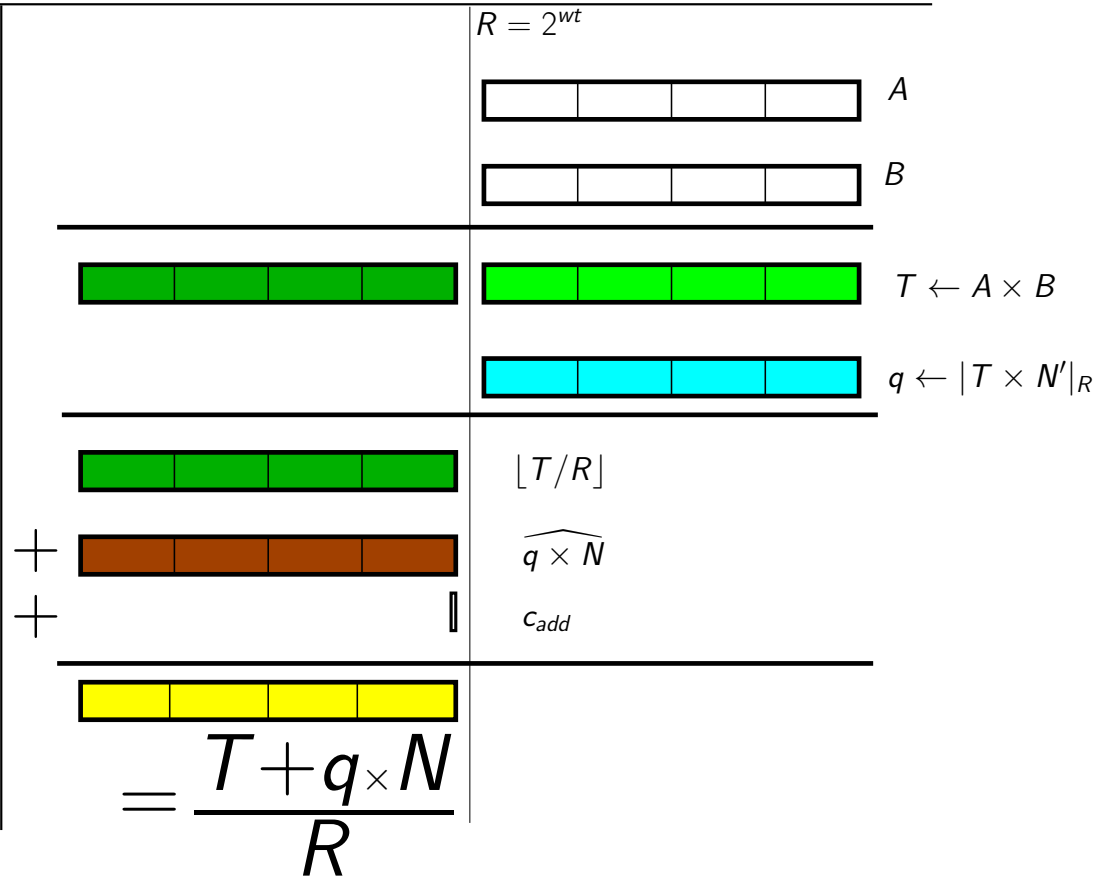
# Algorithm (Scholastique inspired)

**Require:** $T < 4N^2$, $N$ the $wt$-bit modulus, $R = 2^{wt}$,
precomputed value $N' = (-N)^{-1} \bmod R$.
**Ensure:** $C \equiv T \times R^{-1} \bmod N$ and $C < 2N$

$q \leftarrow T \times N' \bmod R$

$\widehat{qN} \leftarrow \lfloor (q \times N)/R \rfloor$
$c_{add} \leftarrow \bigvee_{i=0}^{t_{52}-1} T[i]$
$C \leftarrow \lfloor T/R \rfloor + \widehat{qN} + c_{add}$

**return** $C$



# Proof

## Theorem (Correctness of Truncated Montgomery)

1. Computation of $c_{add}$

Let the *ith* bit of T be its first non-zero least significant bit.

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1\ldots$ |
|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | $x\ldots x$ |

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1$ | |
|---|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | 0 | 1 |
| $q \times N$ | | | | | |
| $T + q \times N$ | $0\ldots0$ | 0 | 0 | 0 | |
| generated carry $c_{add}$ | | | | | |

# Proof

## Theorem (Correctness of Truncated Montgomery)

1. Computation of $c_{add}$

Let the *ith* bit of T be its first non-zero least significant bit.

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1\ldots$ |
|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | $x\ldots x$ |

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1$ | |
|---|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | 0 | 1 |
| $q \times N$ | $0\ldots0$ | 0 | | | |
| $T + q \times N$ | $0\ldots0$ | 0 | 0 | 0 | |
| generated carry $c_{add}$ | $0\ldots0$ | 0 | | | |

# Proof

## Theorem (Correctness of Truncated Montgomery)

1. Computation of $c_{add}$

Let the *ith* bit of T be its first non-zero least significant bit.

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1\ldots$ |
|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | $x\ldots x$ |

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1$ | |
|---|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | 0 | 1 |
| $q \times N$ | $0\ldots0$ | 0 | 1 | | |
| $T + q \times N$ | $0\ldots0$ | 0 | 0 | 0 | |
| generated carry $c_{add}$ | $0\ldots0$ | 0 | 1 | | |

# Proof

## Theorem (Correctness of Truncated Montgomery)

1. Computation of $c_{add}$

Let the *ith* bit of T be its first non-zero least significant bit.

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1\ldots$ |
|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | $x\ldots x$ |

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1$ | |
|---|---|---|---|---|---|
| $T$ | $0\ldots0$ | 0 | 1 | 0 | 1 |
| $q \times N$ | $0\ldots0$ | 0 | 1 | 1 | 0 |
| $T + q \times N$ | $0\ldots0$ | 0 | 0 | 0 | |
| generated carry $c_{add}$ | $0\ldots0$ | 0 | 1 | 1 | |

# Proof

## Theorem (Correctness of Truncated Montgomery)

1. Computation of $c_{add}$

Let the *ith* bit of T be its first non-zero least significant bit.

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1\ldots$ |
|---|---|---|---|---|
| $T$ | $0\ldots0$ | $0$ | $1$ | $x\ldots x$ |

| bit | $1\ldots$ | $i-1$ | $i$ | $i+1$ | |
|---|---|---|---|---|---|
| $T$ | $0\ldots0$ | $0$ | $1$ | $0$ | $1$ |
| $q\times N$ | $0\ldots0$ | $0$ | $1$ | $1$ | $0$ |
| $T+q\times N$ | $0\ldots0$ | $0$ | $0$ | $0$ | |
| generated carry $c_{add}$ | $0\ldots0$ | $0$ | $1$ | $1$ | |

$$c_{add} \leftarrow \bigvee_{i=0}^{t-1} T[i] \qquad (1)$$

# Proof

2. Computation of $\widehat{qN} \leftarrow \lfloor (q \times N)/R \rfloor$

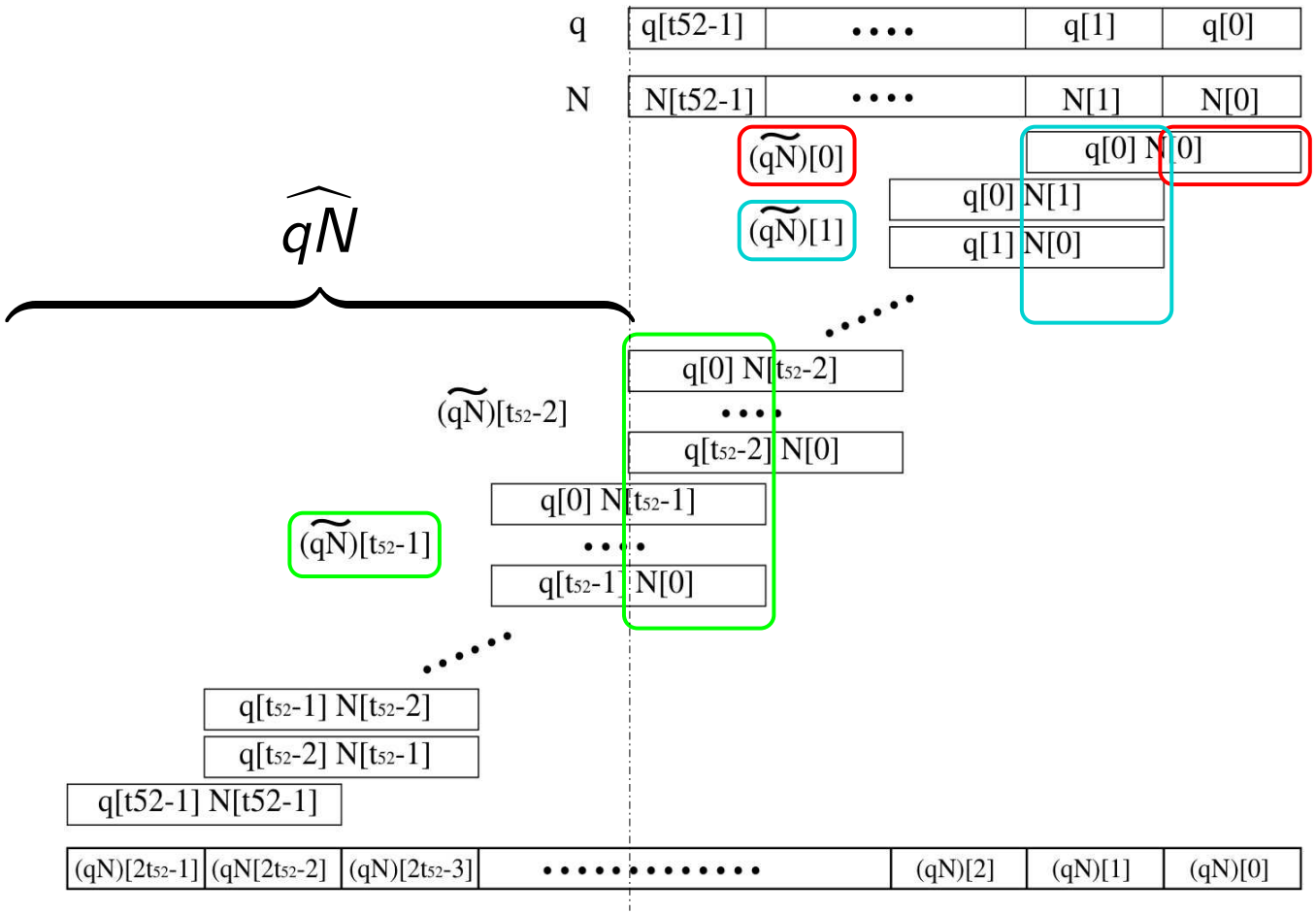



Figure – Detail of $q \times N$ (mots de 52 bits)

# Proof

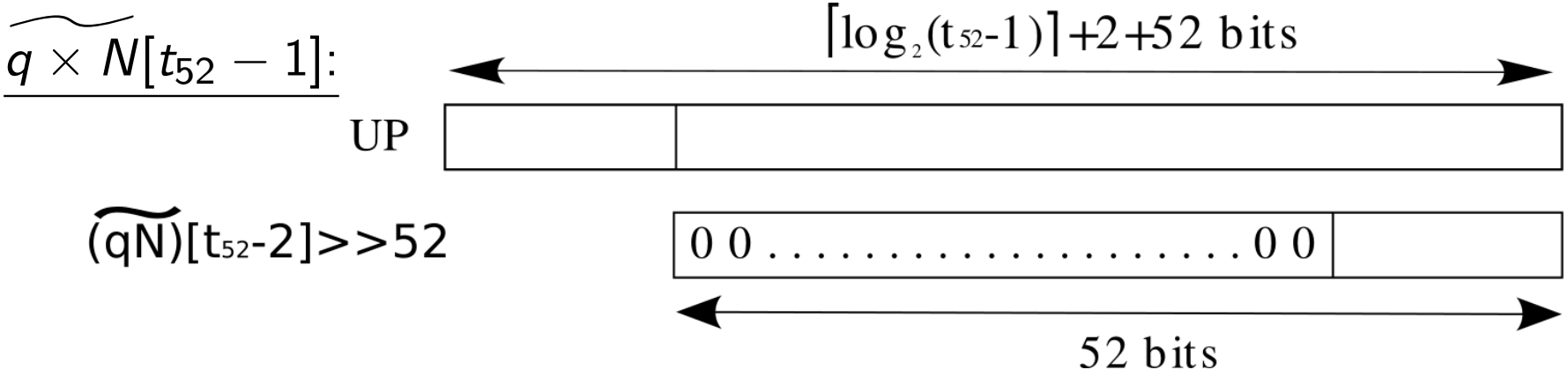


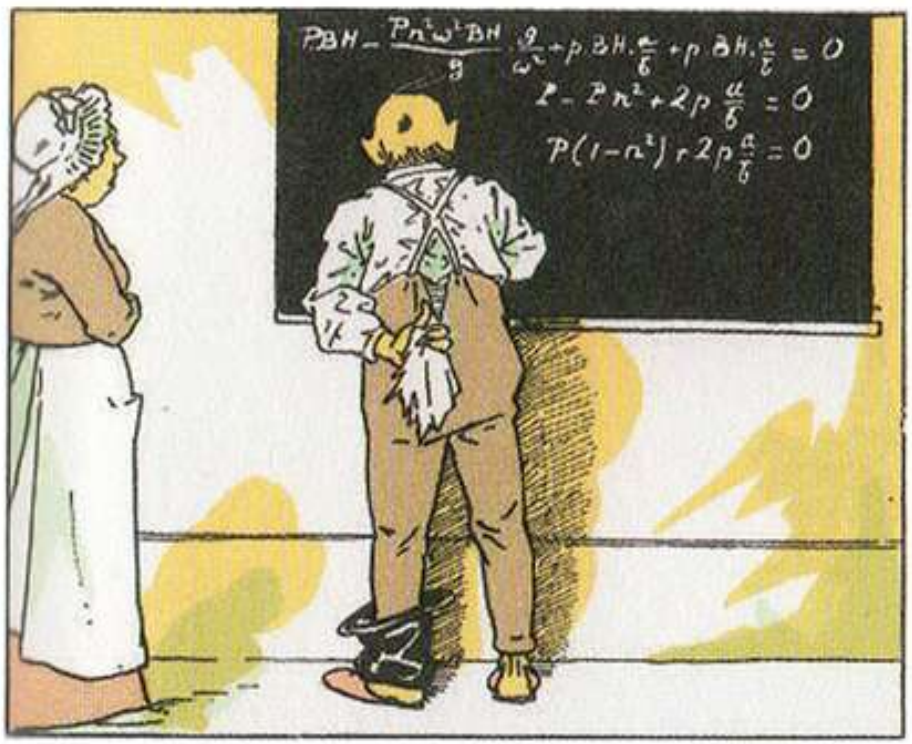

Figure – Detail of $\widetilde{q \times N[t_{52}-1]}$

**The word accumulation in $\widetilde{q \times N[t_{52}-1]}$ yields $UP$.**

- $\rightarrow$ We must add it to the $\widetilde{q \times N[t_{52}]}$ words to yield the first word of $\widehat{qN}$
- The MSBs of $\widetilde{q \times N[t_{52}-2]} >> 52$ are 0s:
  $\rightarrow$ no influence on $UP$

# What about the Complexities?

## What is the gain?



# What about the Complexities?

## Let's consider Schoolbook multiplication:

**Require:** $T < 4N^2$, $N$ the $t$-bit modulus, $R = 2^K$,
precomputed value $N' = (-N)^{-1} \bmod R$.
**Ensure:** $C \equiv T \times R^{-1} \bmod N$ and $C < 2N$

| | |
|---|---|
| $q \leftarrow T \times N' \bmod R$ | $q \leftarrow T \times N' \bmod R$ |
| $C \leftarrow \frac{T+q\times N}{R}$ | $\widehat{qN} \leftarrow \lfloor (q \times N)/R \rfloor$<br>$c_{add} \leftarrow \bigvee_{i=0}^{t_{\mathbf{52}}-1} T[i]$<br>$C \leftarrow \lfloor T/R \rfloor + \widehat{qN} + c_{add}$ |

**return** $C$

# What about the Complexities?

**Let's consider Schoolbook multiplication:**

**Require:** $T < 4N^2$, $N$ the $t$-bit modulus, $R = 2^K$,
precomputed value $N' = (-N)^{-1} \bmod R$.
**Ensure:** $C \equiv T \times R^{-1} \bmod N$ and $C < 2N$

1 SB multiplication to get $T \leftarrow A \times B$

| | |
|---|---|
| $q \leftarrow T \times N' \bmod R$ | $q \leftarrow T \times N' \bmod R$ |
| 0.5 SB multiplication | 0.5 SB multiplication |
| $C \leftarrow \frac{T+q\times N}{R}$ | $\widehat{qN} \leftarrow \lfloor (q \times N)/R \rfloor$<br>$c_{add} \leftarrow \bigvee_{i=0}^{t_{52}-1} T[i]$<br>$C \leftarrow \lfloor T/R \rfloor + \widehat{qN} + c_{add}$ |
| 1 SB multiplication to get $q \times N$ | 0.5 SB multiplication to get $\widehat{qN}$ |

**return** $C$

| | |
|---|---|
| Total: 2.5 SB mult. | 2.0 SB mult. |

# What about Karatsuba Multiplication?

Size operand of $t$ bits. One splits $A$ and $B$ in two halves

$$A = a_\ell + 2^{t/2} a_h, B = b_\ell + 2^{t/2} b_h.$$

$A \times B$ is expressed in terms of **three multiplications** of half size:

$$\begin{cases} D_0 = D_{0\ell} + 2^{t/2} D_{0h} \leftarrow & a_\ell \times b_l, \\ D_1 = D_{1\ell} + 2^{t/2} D_{1h} \leftarrow & (a_\ell + a_h) \times (b_\ell + b_h), \\ D_2 = D_{2\ell} + 2^{t/2} D_{2h} \leftarrow & a_h \times b_h. \end{cases}$$

$$\begin{aligned} A \times B = \quad & D_{0\ell} \\ & + 2^{t/2}(D_{0h} + D_{1\ell} - D_{0\ell} - D_{2\ell}) \\ & + 2^{t}(D_{2\ell} + D_{1h} - D_{0h} - D_{2h}) \\ & + 2^{3t/2} D_{2h}. \end{aligned}$$

# What about Karatsuba Multiplication?

Size operand of $t$ bits. One splits $A$ and $B$ in two halves

$$A = a_\ell + 2^{t/2} a_h, B = b_\ell + 2^{t/2} b_h.$$

$A \times B$ is expressed in terms of **three multiplications** of half size:

$$\begin{cases} D_0 = D_{0\ell} + 2^{t/2} D_{0h} \leftarrow & a_\ell \times b_l, \\ D_1 = D_{1\ell} + 2^{t/2} D_{1h} \leftarrow & (a_\ell + a_h) \times (b_\ell + b_h), \\ D_2 = D_{2\ell} + 2^{t/2} D_{2h} \leftarrow & a_h \times b_h. \end{cases}$$

$$\begin{aligned} A \times B = \quad & D_{0\ell} \\ & + 2^{t/2}(D_{0h} + D_{1\ell} - D_{0\ell} - D_{2\ell}) \\ & + 2^{t}(D_{2\ell} + D_{1h} - D_{0h} - D_{2h}) \\ & + 2^{3t/2} D_{2h}. \end{aligned}$$

$$(A \times B)_{lo} = \quad D_{0\ell} + 2^{t/2}(D_{0h} + D_{1\ell} - D_{0\ell} - D_{2\ell}). \quad \rightarrow 4/6$$

# What about Karatsuba Multiplication?

Size operand of $t$ bits. One splits $A$ and $B$ in two halves

$$A = a_\ell + 2^{t/2} a_h, B = b_\ell + 2^{t/2} b_h.$$

$A \times B$ is expressed in terms of **three multiplications** of half size:

$$\begin{cases} D_0 = D_{0\ell} + 2^{t/2} D_{0h} \leftarrow & a_\ell \times b_l, \\ D_1 = D_{1\ell} + 2^{t/2} D_{1h} \leftarrow & (a_\ell + a_h) \times (b_\ell + b_h), \\ D_2 = D_{2\ell} + 2^{t/2} D_{2h} \leftarrow & a_h \times b_h. \end{cases}$$

$$\begin{aligned} A \times B = \; & D_{0\ell} \\ & + 2^{t/2}(D_{0h} + D_{1\ell} - D_{0\ell} - D_{2\ell}) \\ & + 2^{t}(D_{2\ell} + D_{1h} - D_{0h} - D_{2h}) \\ & + 2^{3t/2} D_{2h}. \end{aligned}$$

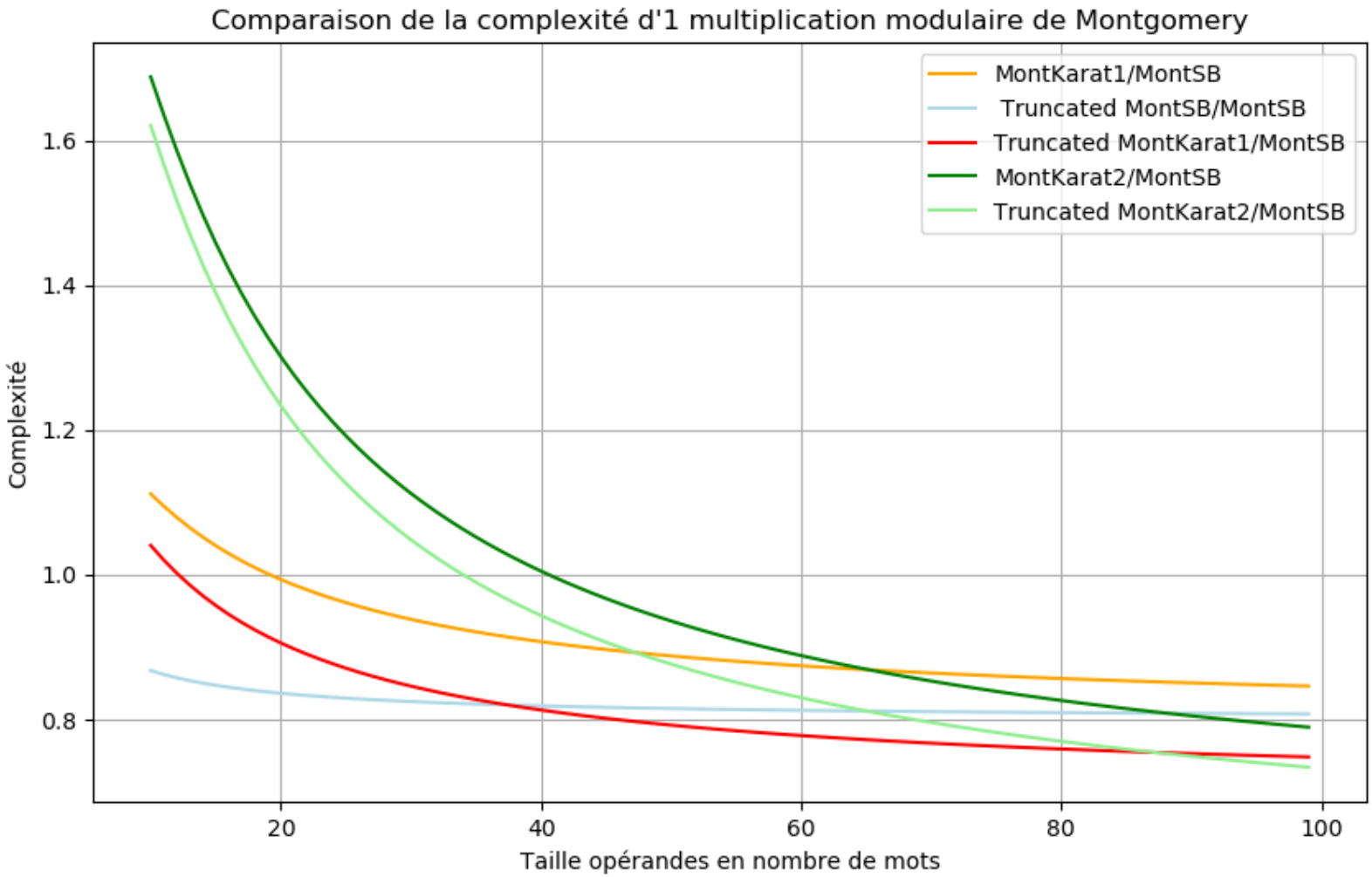
$$(A \times B)_{hi} = \; 2^{t}(D_{2\ell} + D_{1h} - D_{0h} - D_{2h}) + 2^{3t/2} D_{2h}. \quad \rightarrow 4/6$$

# Complexities

| Montgomery Multiplication | | | |
|---|---|---|---|
| type | | $\#\mathcal{M}$ | gain tr/nml |
| Schoolbook | normal | 2.5 | 0.5, 20 % |
| | truncated | 2.0 | |
| Karatsuba | normal | 2.67 | 0.33, 12.5 % |
| | truncated | 2.33 | |
| Karatsuba 2 | normal | 2.78 | 0.22, 8 % |
| | truncated | 2.56 | |

Complexity in full multiprecision multiplications.

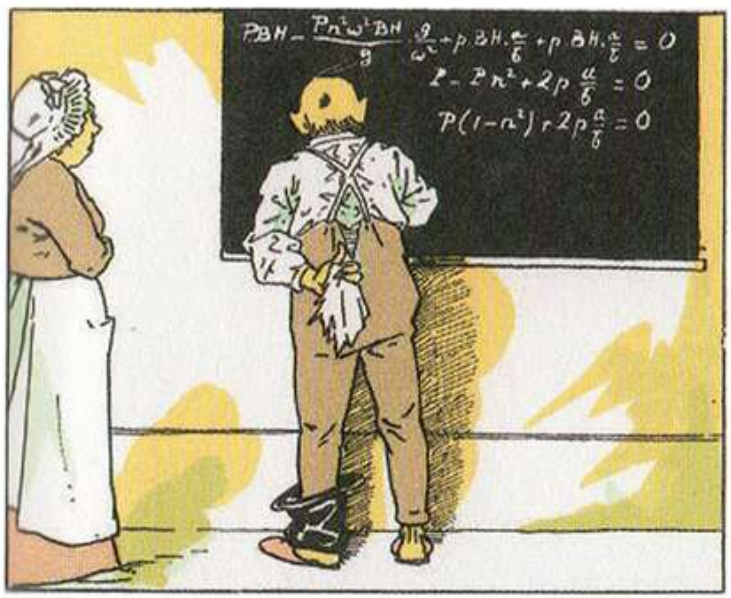
# Complexities




(1040) (2080) (4160) → 52 bit words

# Let's implement!

What about software implementation of Scholastique inspired approach?



# Batch software computations AVX512

Modern Intel processors : 32 zmm registers of size 512 bits

**zmm of size 512 bits:**

i.e. $8 \times 64$ bits, $\rightarrow \{a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0\}$,
$a_0$, $a_1$, $a_2$ , $a_3$, $a_4$, $a_5$, $a_6$ and $a_7$, quadwords (i.e. 64-bit)

Single Instruction Multiple Data (SIMD) instruction set : one single instruction processes eight contiguous 64-bit words (or 16 32-bit words...).

# VPMADD52 instruction

Instruction available since the IceLake Intel processor. C-intrinsic:

`__m512i _mm512_madd52lo_epu64 (__m512i a, __m512i b, __m512i c)`

**for** $i = 0 \dots 7$ **do**
  $Dest_i \leftarrow a_i + [(b_i * c_i) \bmod 2^{52}]$ // lowest 52 bits of $b * c$
**end for**

`__m512i _mm512_madd52hi_epu64 (__m512i a, __m512i b, __m512i c)`

**for** $i = 0 \dots 7$ **do**
  $Dest_i \leftarrow a_i + [(b_i * c_i) >> 52]$ // highest 52 bits of $b * c$
**end for**

$\rightarrow$ Latency : 4, Throughput : $0,5$

To use this instruction we need to take $w = 52$ instead of 64.

# Multiplication : Idea

## We set

$$A_i = \sum_{j=0}^{t} a_{ij} 2^{64*j} = \sum_{j=0}^{t_{52}} a52_{ij} 2^{52*j}$$

## and store the values as follows:

| | 64 bits / 52 bits | | 64 bits / 52 bits | | 64 bits / 52 bits | | 64 bits / 52 bits | | 64 bits / 52 bits | | 64 bits / 52 bits | | 64 bits / 52 bits | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A512_0$ | 0 | $a52_{00}$ | 0 | $a52_{10}$ | 0 | $a52_{20}$ | 0 | $a52_{30}$ | 0 | $a52_{40}$ | 0 | $a52_{50}$ | 0 | $a52_{60}$ | 0 | $a52_{70}$ |
| $A512_1$ | 0 | $a52_{01}$ | 0 | $a52_{11}$ | 0 | $a52_{21}$ | 0 | $a52_{31}$ | 0 | $a52_{41}$ | 0 | $a52_{51}$ | 0 | $a52_{61}$ | 0 | $a52_{71}$ |
| | | | | | | | | ⋮ | | | | | | | | |
| $t_{52}$ 512 bit lines | | | | | | | | | | | | | | | | |

Word-slicing storage of eight values, in radix $2^{52}$
in order to use vectorized 52 bit FMAs (VPMADD52).

# Performances

**Batch Montgomery multiplications (normal and truncated) # clock cycles.**

| | | Montgomery modular multiplications | | |
|---|---|---|---|---|
| Modulus Size | | 1024 | 2048 | 4096 |
| GMP (×8) | | 9862 | 35497 | 120342 |
| OpenSSL (AVX2) (×8) | | 7949 | 29928 | 116898 |
| **Batch** | **Multiplication** | **This work** | | |
| | Schoolbook | 2276 | 8696 | 38609 |
| -20 % | Truncated Schoolbook | **1847** | 7306 | 30492 |
| | speedup vs GMP | 5.34 | 4.86 | 3.95 |
| | speedup vs OpenSSL | 4.31 | 4.10 | 3.69 |
| | Karatsuba | 2423 | 8400 | 34628 |
| -12 to -20 % | Truncated Karatsuba | - | **7286** | **27594** |
| | speedup vs GMP | 4.07 | 4.87 | 4.36 |
| | speedup vs OpenSSL | 3.28 | 4.12 | 4.24 |

# Performances

**Batch Montgomery squarings (normal and truncated), # clock cycles.**

| | | Montgomery modular squarings | | |
|---|---|---|---|---|
| | Modulus Size | 1024 | 2048 | 4096 |
| | GMP ($\times 8$) | 8358 | 30465 | 101373 |
| | **Batch** **Multiplication** | **This work** | | |
| | Schoolbook | 1885 | 7154 | 32107 |
| -20 % | Truncated Schoolbook | **1439** | **5548** | **23413** |
| | speedup vs GMP | 5.81 | 5.49 | 4.33 |
| -12 to | Karatsuba | 2191 | 7330 | 28841 |
| -20 % | Truncated Karatsuba | - | 6202 | 23736 |
| | speedup vs GMP | 3.81 | 4.91 | 4.27 |

# Performances

Batch of 8 modular fixed-window exponentiations, constant time, $\#10^3$ clock cycles.

| $\times 8$ **Modular Fixed-Window exponentiation ( $\times 10^3$#cc)** | | | |
|---|---|---|---|
| Modulus Size | 1024 | 2048 | 4096 |
| `GMP` ($\times 8$) | 11333 | 81288 | 614637 |
| `OpenSSL BN_mod_exp_mont_consttime (AVX2)` ($\times 8$) | 8313 | 58445 | 434359 |
| `OpenSSL BN_mod_exp_mont_consttimex2 (AVX512)` ($\times 4$) | 3648 | 21674 | 442533 |
| **Batch multiplication** | **This work** | | |
| Schoolbook | 2589 | 19678 | 177095 |
| -20 % Truncated Schoolbook | **2090** | **15736** | 131753 |
| speedup vs `GMP` | 5.42 | 5.17 | 4.66 |
| speedup vs `OpenSSL BN_mod_exp_mont_consttime` | 3.98 | 3.71 | 3.30 |
| speedup vs `OpenSSL BN_mod_exp_mont_consttimex2` | 1.75 | 1.38 | - |
| Karatsuba | - | 19717 | 144567 |
| $\approx$ -10 % Truncated Karatsuba | - | 17124 | **129015** |
| speedup vs `GMP` | - | 4.75 | 4.76 |
| speedup vs `OpenSSL BN_mod_exp_mont_consttime` | - | 3.41 | 3.37 |
| speedup vs texttt`OpenSSL BN_mod_exp_mont_consttimex2` | - | 1.27 | - |

# Conclusion

## Truncated Montgomery

- Schoolbook : gain of 20%
- Karatsuba : gain of 12.5%
- Karatsuba 2 : gain of 8%

## Batch Vectorized Implantation (1024 to 4096 bits)

- Exponentiation (with Truncated)
  - Gain between 4.66 and 5.42 vs GMP
  - Gain between 3.30 and 3.98 vs OpenSSL
  - Gain between 1.38 and 1.75 vs `BN_mod_exp_mont_consttimex2` (1024 and 2048 bits)
    $\Rightarrow$ similar with multiplication or squaring
- Truncated Montgomery
  - Gain of about 20% in Schoolb. and 10% to 20 % in Karat.!

# Questions ?

Thank you very much for your attention!