High-performance Elliptic Curve Cryptography by Using the CIOS Method for Modular Multiplication

A*mine Mrabet*, Nadia El-Mrabet, **Ronan Lashermes**, Jean-Baptiste Rigaud, Belgacem Bouallegue, Sihem Mesnager and Mohsen Machhout

September 2016

Introduction

- Public key cryptography is still costly (computing resources).
- Elliptic Curve Cryptography has a better cost/security trade-off w.r.t. RSA.
- We can still reduce the cost with better **hardware architectures**.

Elliptic Curve Cryptography (ECC)

Why?

Elliptic curves allow to define groups with a hard Discrete Logarithm Problem. In the general case, cracking methods are far less efficient than for RSA.

Elliptic Curve Cryptography (ECC)

Why?

Elliptic curves allow to define groups with a hard Discrete Logarithm Problem. In the general case, cracking methods are far less efficient than for RSA.

How? (simplified)

Let $p > 3$ a big prime, $E(\mathbb{F}_p)$ is the (short Weierstrass) elliptic curve

$$E(\mathbb{F}_p) : y^2 = x^3 + ax + b,$$

where $x, y, a, b \in \mathbb{F}_p$ with $4a^3 + 27b^2 \neq 0$.

EC Group

The points (x, y) on the curve define an abelian group together with the **point at infinity** 0_∞, the neutral element for addition.

EC Group

The points (x, y) on the curve define an abelian group together with the **point at infinity** 0_∞, the neutral element for addition.

Jacobian coordinates

The triple $(x : y : z)$ can be mapped to $(x/z^2, y/z^3)$ if $z \neq 0$. If $z = 0$ it is 0_∞. The curve becomes: $y^2 = x^3 + axz^4 + bz^6$.

Operations in Jacobian coordinates ($a = 0$, points $\neq 0_\infty$)

Doubling (7S+5M+13A)

$T(X_T : Y_T : Z_T) = 2 \cdot Q(X_Q : Y_Q : Z_Q)$.
$X_T = 9X_Q^4 - 8X_Q Y_Q^2$,
$Y_T = 3X_Q^2(4X_Q Y_Q - X_T) - 8Y_Q^4$,
$Z_T = 2Y_Q Z_Q$.

Operations in Jacobian coordinates ($a = 0$, points $\neq 0_\infty$)

Doubling (7S+5M+13A)

$T(X_T : Y_T : Z_T) = 2 \cdot Q(X_Q : Y_Q : Z_Q)$.
$X_T = 9X_Q^4 - 8X_Q Y_Q^2$,
$Y_T = 3X_Q^2(4X_Q Y_Q - X_T) - 8Y_Q^4$,
$Z_T = 2Y_Q Z_Q$.

Addition (4S + 14M + 6A)

$R = T + Q$.
$X_R = (2Y_Q Z_T^3 - 2Y_T)^2 - 4(X_Q Z_T^2 - X_T)^3 - 8(X_Q Z_T^2 - X_T)^2 X_T$,
$Y_R =$
$(2Y_Q Z_T^3 - 2Y_T)(4X_T(X_Q Z_T^2 - X_T) - X_R) - 8Y_T(X_Q Z_T^2 - X_T)^3$,
$Z_R = 2Z_T(X_Q Z_T^2 - X_T)$.

Montgomery Modular Multiplication (MMM)

MMM

MMM provides an efficient way for modular multiplication $\mod p$ (noted $\cdot$): there is no division by p.

Montgomery Modular Multiplication (MMM)

MMM

MMM provides an efficient way for modular multiplication $\mod p$ (noted $\cdot$): there is no division by p.

Residue

Let $a, b, R \in \mathbb{F}_p$ where R is Montgomery's residue.
$a' = aR \mod p$ is said to be a in Montgomery's form.
$a \cdot b = abR^{-1} \mod p$, as a consequence
$a' \cdot b' = aRbRR^{-1} \mod p = abR \mod p = (ab)'$.

Montgomery Modular Multiplication (MMM)

MMM

MMM provides an efficient way for modular multiplication $\mod p$ (noted $\cdot$): there is no division by p.

Residue

Let $a, b, R \in \mathbb{F}_p$ where R is Montgomery's residue.
$a' = aR \mod p$ is said to be a in Montgomery's form.
$a \cdot b = abR^{-1} \mod p$, as a consequence
$a' \cdot b' = aRbRR^{-1} \mod p = abR \mod p = (ab)'$.

Conversion

Field values are converted in Montgomery's form at the beginning of the computation and back to normal at the end.

How to compute MMM?

Koç's multiword CIOS algorithm

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $\quad C \leftarrow 0$;
4 $\quad$ **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $\quad\quad (C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$
6 $\quad\quad T[j] \leftarrow S$
7 $\quad (C, S) \leftarrow T[s] + C$
8 $\quad T[s] \leftarrow S$
9 $\quad T[s + 1] \leftarrow C$
10 $\quad m \leftarrow T[0] \cdot p' \bmod 2^w$
11 $\quad (C, S) \leftarrow T[0] + m \cdot p[0]$
12 $\quad$ **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $\quad\quad (C, S) \leftarrow T[j] + m \cdot p[j] + C$
14 $\quad\quad T[j - 1] \leftarrow S$
15 $\quad (C, S) \leftarrow T[s] + C$
16 $\quad T[s - 1] \leftarrow S$
17 $\quad T[s] \leftarrow T[s + 1] + C$
18 **return** T;

CIOS details

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $\quad C \leftarrow 0$;
4 $\quad$ **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $\quad\quad (C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$
6 $\quad\quad T[j] \leftarrow S$
7 $\quad (C, S) \leftarrow T[s] + C$
8 $\quad T[s] \leftarrow S$
9 $\quad T[s + 1] \leftarrow C$
10 $\quad m \leftarrow T[0] \cdot p' \bmod 2^w$
11 $\quad (C, S) \leftarrow T[0] + m \cdot p[0]$
12 $\quad$ **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $\quad\quad (C, S) \leftarrow T[j] + m \cdot p[j] + C$
14 $\quad\quad T[j - 1] \leftarrow S$
15 $\quad (C, S) \leftarrow T[s] + C$
16 $\quad T[s - 1] \leftarrow S$
17 $\quad T[s] \leftarrow T[s + 1] + C$
18 **return** T;

Multiplication

Reduction

Benefits

- Low memory footprint,
- apart from some precomputations (p', R...), easy to change p and operand sizes,
- neat structure, without divisions, easy to implement in hardware.

Basics

Here, each operation takes 1 unit of time.
Let's compute $r = a \cdot b + b + c$.

Sequential

Time	$\cdot$	$+$	Operations
1	x		$t1 = a \cdot b$
2		x	$t2 = b + c$
3		x	$r = t1 + t2$

Basics

Here, each operation takes 1 unit of time.
Let's compute $r = a \cdot b + b + c$.

Sequential

Time	$\cdot$	$+$	Operations
1	x		$t1 = a \cdot b$
2		x	$t2 = b + c$
3		x	$r = t1 + t2$

Parallel

Time	$\cdot$	$+$	Operations
1	x	x	$t1 = a \cdot b,\ t2 = b + c$
2		x	$r = t1 + t2$

Basics - 2

Here, each operation takes 1 unit of time.
Let's compute $r = a \cdot b + b + c$.

Atomic

Latency	Throughput	$\cdot$	$+$	Operations
2	0.5	1	2	$r = a \cdot b + b + c$

The choice of operations and how they are chained together is called **scheduling**.

Basics - 2

Here, each operation takes 1 unit of time.
Let's compute $r = a \cdot b + b + c$.

Atomic

Latency	Throughput	$\cdot$	$+$	Operations
2	0.5	1	2	$r = a \cdot b + b + c$

Pipelined

Latency	Throughput	$\cdot$	$+$	Operations
$2 + \epsilon$	0.5	1	1	$1 : t1 = a \cdot b, t2 = b + c, 2 : r = t1 + t2$
$2 + \epsilon$	1	1	2	$1 : t1 = a \cdot b, t2 = b + c, 2 : r = t1 + t2$

The choice of operations and how they are chained together is called **scheduling**.

Systolic arrays

A systolic array is an architecture both parallel and pipelined.
To create such an architecture, we have to identify small **Processing Elements (PEs)** (no control flow logic).

Where is ~~Waldo~~ the PE?

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $C \leftarrow 0$;
4 **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $(C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$
6 $T[j] \leftarrow S$
7 $(C, S) \leftarrow T[s] + C$
8 $T[s] \leftarrow S$
9 $T[s + 1] \leftarrow C$
10 $m \leftarrow T[0] \cdot p' \bmod 2^w$
11 $(C, S) \leftarrow T[0] + m \cdot p[0]$
12 **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $(C, S) \leftarrow T[j] + m \cdot p[j] + C$
14 $T[j - 1] \leftarrow S$
15 $(C, S) \leftarrow T[s] + C$
16 $T[s - 1] \leftarrow S$
17 $T[s] \leftarrow T[s + 1] + C$
18 **return** T;

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $\quad C \leftarrow 0$;
4 $\quad$ **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $\quad\quad (C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$
6 $\quad\quad T[j] \leftarrow S$
7 $\quad (C, S) \leftarrow T[s] + C$
8 $\quad T[s] \leftarrow S$
9 $\quad T[s + 1] \leftarrow C$
10 $\quad m \leftarrow T[0] \cdot p' \bmod 2^w$
11 $\quad (C, S) \leftarrow T[0] + m \cdot p[0]$
12 $\quad$ **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $\quad\quad (C, S) \leftarrow T[j] + m \cdot p[j] + C$
14 $\quad\quad T[j - 1] \leftarrow S$
15 $\quad (C, S) \leftarrow T[s] + C$
16 $\quad T[s - 1] \leftarrow S$
17 $\quad T[s] \leftarrow T[s + 1] + C$
18 **return** T;

α

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $\quad C \leftarrow 0$;
4 $\quad$ **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $\quad\quad (C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$ α
6 $\quad\quad T[j] \leftarrow S$
7 $\quad (C, S) \leftarrow T[s] + C$
8 $\quad T[s] \leftarrow S$ α_f
9 $\quad T[s + 1] \leftarrow C$
10 $\quad m \leftarrow T[0] \cdot p' \bmod 2^w$
11 $\quad (C, S) \leftarrow T[0] + m \cdot p[0]$
12 $\quad$ **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $\quad\quad (C, S) \leftarrow T[j] + m \cdot p[j] + C$
14 $\quad\quad T[j - 1] \leftarrow S$
15 $\quad (C, S) \leftarrow T[s] + C$
16 $\quad T[s - 1] \leftarrow S$
17 $\quad T[s] \leftarrow T[s + 1] + C$
18 **return** T;

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $\quad C \leftarrow 0$;
4 $\quad$ **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $\quad\quad (C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$ — α
6 $\quad\quad T[j] \leftarrow S$
7 $\quad (C, S) \leftarrow T[s] + C$ — α_f
8 $\quad T[s] \leftarrow S$
9 $\quad T[s + 1] \leftarrow C$
10 $\quad m \leftarrow T[0] \cdot p' \bmod 2^w$ — β
11 $\quad (C, S) \leftarrow T[0] + m \cdot p[0]$
12 $\quad$ **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $\quad\quad (C, S) \leftarrow T[j] + m \cdot p[j] + C$
14 $\quad\quad T[j - 1] \leftarrow S$
15 $\quad (C, S) \leftarrow T[s] + C$
16 $\quad T[s - 1] \leftarrow S$
17 $\quad T[s] \leftarrow T[s + 1] + C$
18 **return** T;

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $\quad C \leftarrow 0$;
4 $\quad$ **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $\quad\quad (C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$ — α
6 $\quad\quad T[j] \leftarrow S$
7 $\quad (C, S) \leftarrow T[s] + C$
8 $\quad T[s] \leftarrow S$ — α_f
9 $\quad T[s + 1] \leftarrow C$
10 $\quad m \leftarrow T[0] \cdot p' \bmod 2^w$
11 $\quad (C, S) \leftarrow T[0] + m \cdot p[0]$ — β
12 $\quad$ **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $\quad\quad (C, S) \leftarrow T[j] + m \cdot p[j] + C$ — γ
14 $\quad\quad T[j - 1] \leftarrow S$
15 $\quad (C, S) \leftarrow T[s] + C$
16 $\quad T[s - 1] \leftarrow S$
17 $\quad T[s] \leftarrow T[s + 1] + C$
18 **return** T;

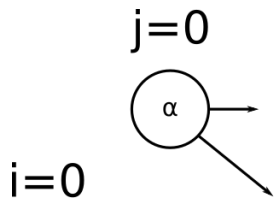
γ_f

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0;
2  for i ← 0 to s − 1 do
3      C ← 0;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C                    α
6          T[j] ← S
7      (C, S) ← T[s] + C                                      α_f
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p' mod 2^w                                  β
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do
13         (C, S) ← T[j] + m · p[j] + C                       γ
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                                      γ_f
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T;
```

S=8, Time=1

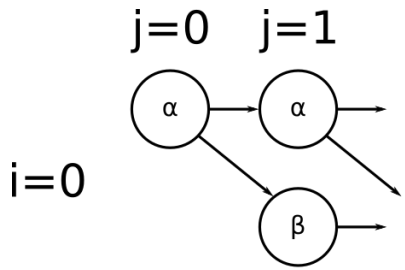


Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0;
2  for i ← 0 to s − 1 do
3      C ← 0;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C                    α
6          T[j] ← S
7      (C, S) ← T[s] + C                                      αf
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p′ mod 2^w                                  β
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do
13         (C, S) ← T[j] + m · p[j] + C                       γ
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                                      γf
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T;
```

S=8, Time=2

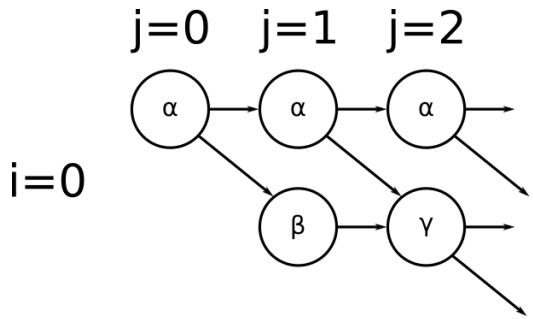


Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0;
2  for i ← 0 to s − 1 do
3      C ← 0;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C                    α
6          T[j] ← S
7      (C, S) ← T[s] + C                                      αf
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p′ mod 2^w                                  β
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do
13         (C, S) ← T[j] + m · p[j] + C                       γ
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                                      γf
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T;
```

S=8, Time=3

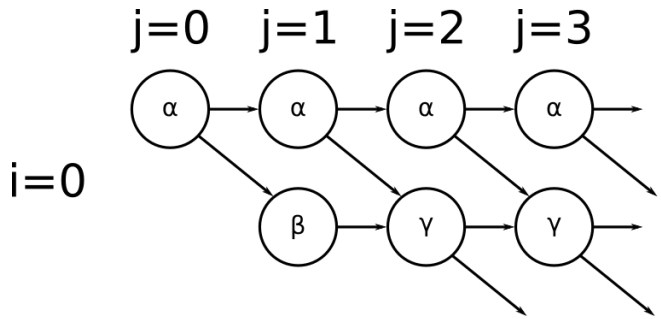


Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

1 $T \leftarrow 0$;
2 **for** $i \leftarrow 0$ ***to*** $s - 1$ **do**
3 $\quad C \leftarrow 0$;
4 $\quad$ **for** $j \leftarrow 0$ ***to*** $s - 1$ **do**
5 $\quad\quad (C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$ — α
6 $\quad\quad T[j] \leftarrow S$
7 $\quad (C, S) \leftarrow T[s] + C$ — α_f
8 $\quad T[s] \leftarrow S$
9 $\quad T[s + 1] \leftarrow C$
10 $\quad m \leftarrow T[0] \cdot p' \bmod 2^w$ — β
11 $\quad (C, S) \leftarrow T[0] + m \cdot p[0]$
12 $\quad$ **for** $j \leftarrow 1$ ***to*** $s - 1$ **do**
13 $\quad\quad (C, S) \leftarrow T[j] + m \cdot p[j] + C$ — γ
14 $\quad\quad T[j - 1] \leftarrow S$
15 $\quad (C, S) \leftarrow T[s] + C$ — γ_f
16 $\quad T[s - 1] \leftarrow S$
17 $\quad T[s] \leftarrow T[s + 1] + C$
18 **return** T;

S=8, Time=4

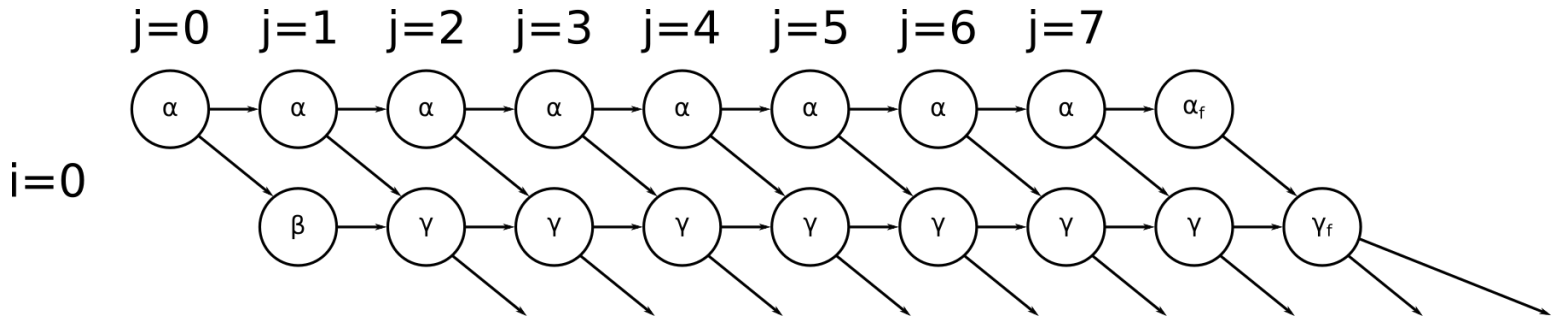


Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0;
2  for i ← 0 to s − 1 do
3      C ← 0;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C                    α
6          T[j] ← S
7      (C, S) ← T[s] + C                                      αf
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p′ mod 2^w                                  β
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do
13         (C, S) ← T[j] + m · p[j] + C                       γ
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                                      γf
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T;
```

S=8, Time=10

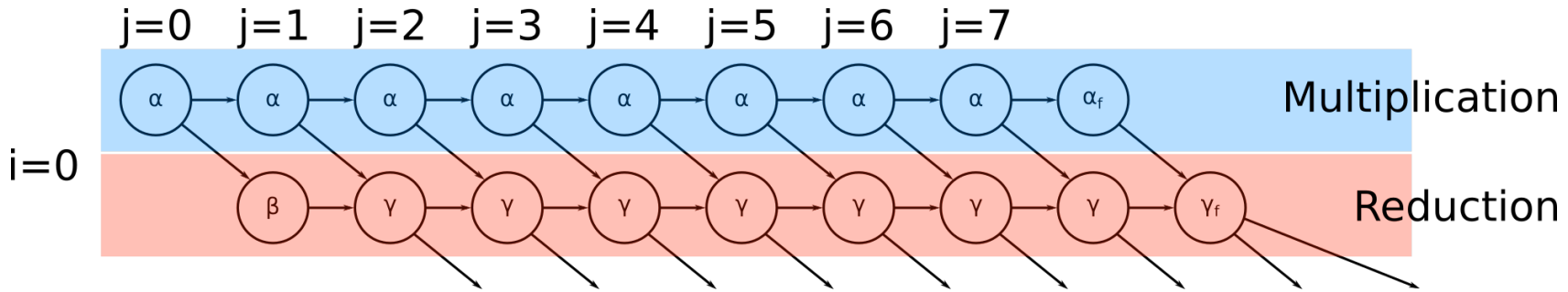


Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0;
2  for i ← 0 to s − 1 do
3      C ← 0;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C                    α
6          T[j] ← S
7      (C, S) ← T[s] + C                                      αf
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p′ mod 2^w                                  β
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do
13         (C, S) ← T[j] + m · p[j] + C                       γ
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                                      γf
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T;
```

S=8, Time=10

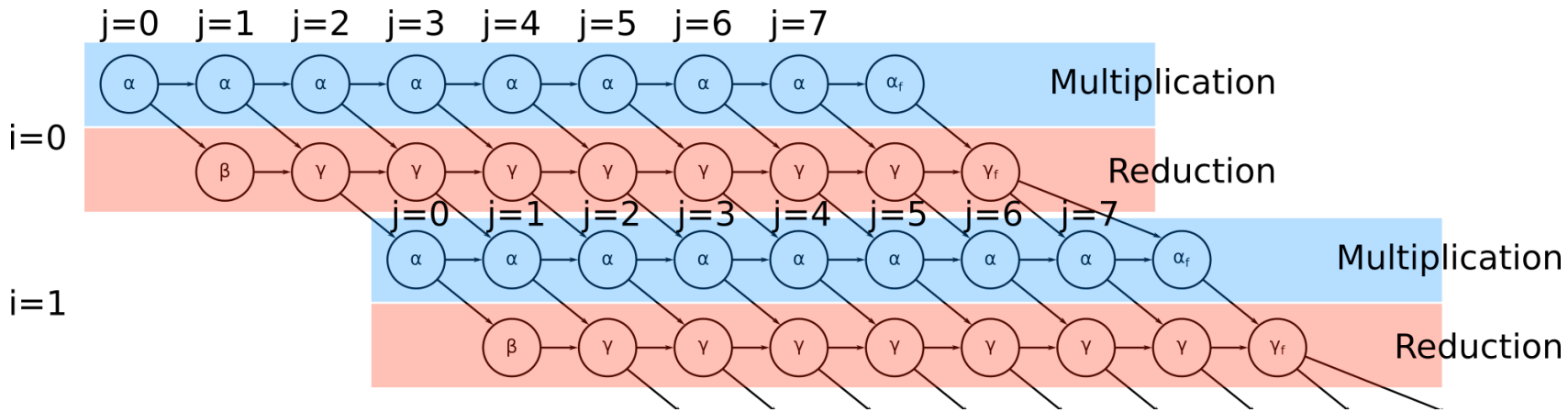


Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0;
2  for i ← 0 to s − 1 do
3      C ← 0;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C                    α
6          T[j] ← S
7      (C, S) ← T[s] + C                                      αf
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p′ mod 2^w                                  β
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do
13         (C, S) ← T[j] + m · p[j] + C                       γ
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                                      γf
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T;
```

S=8, Time=13

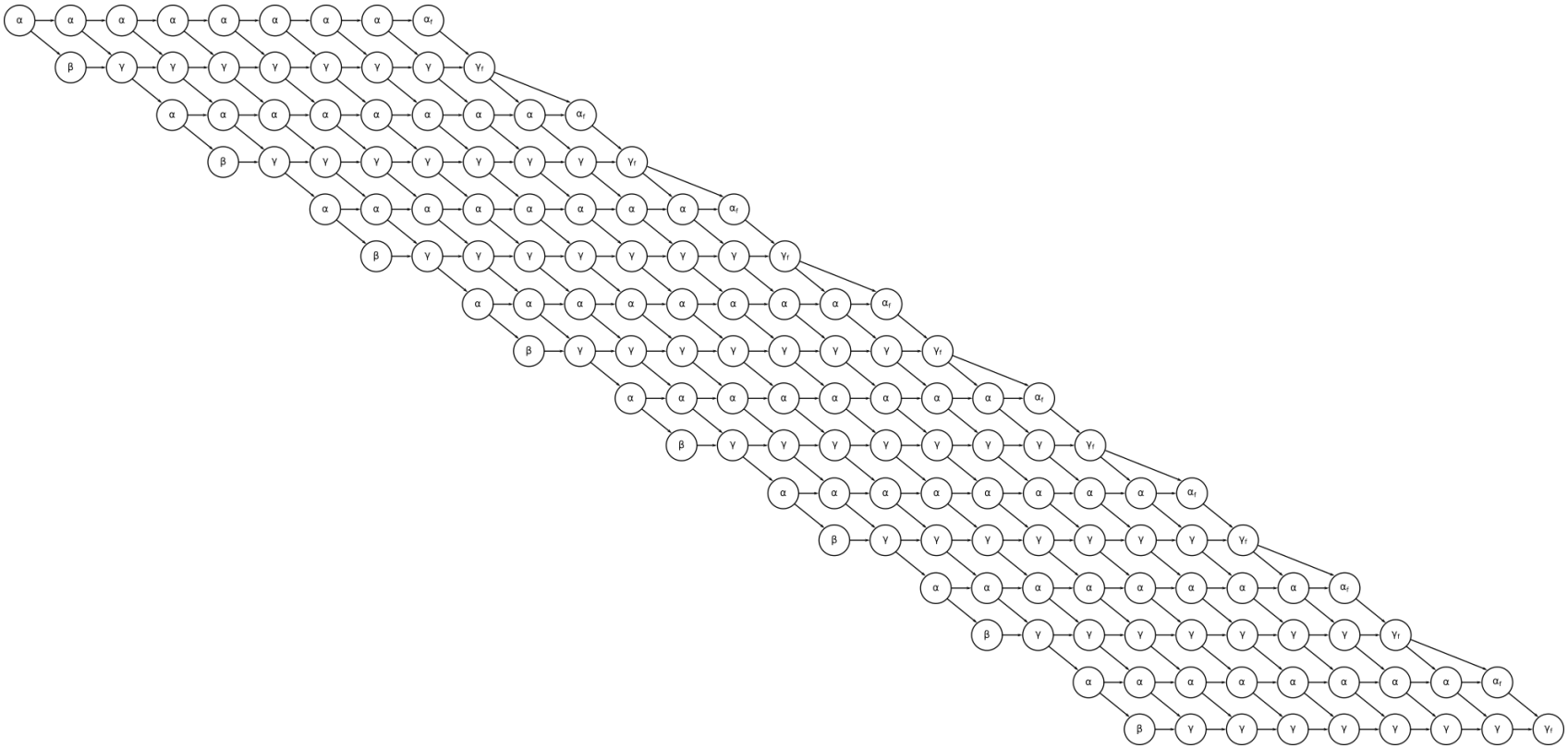


Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

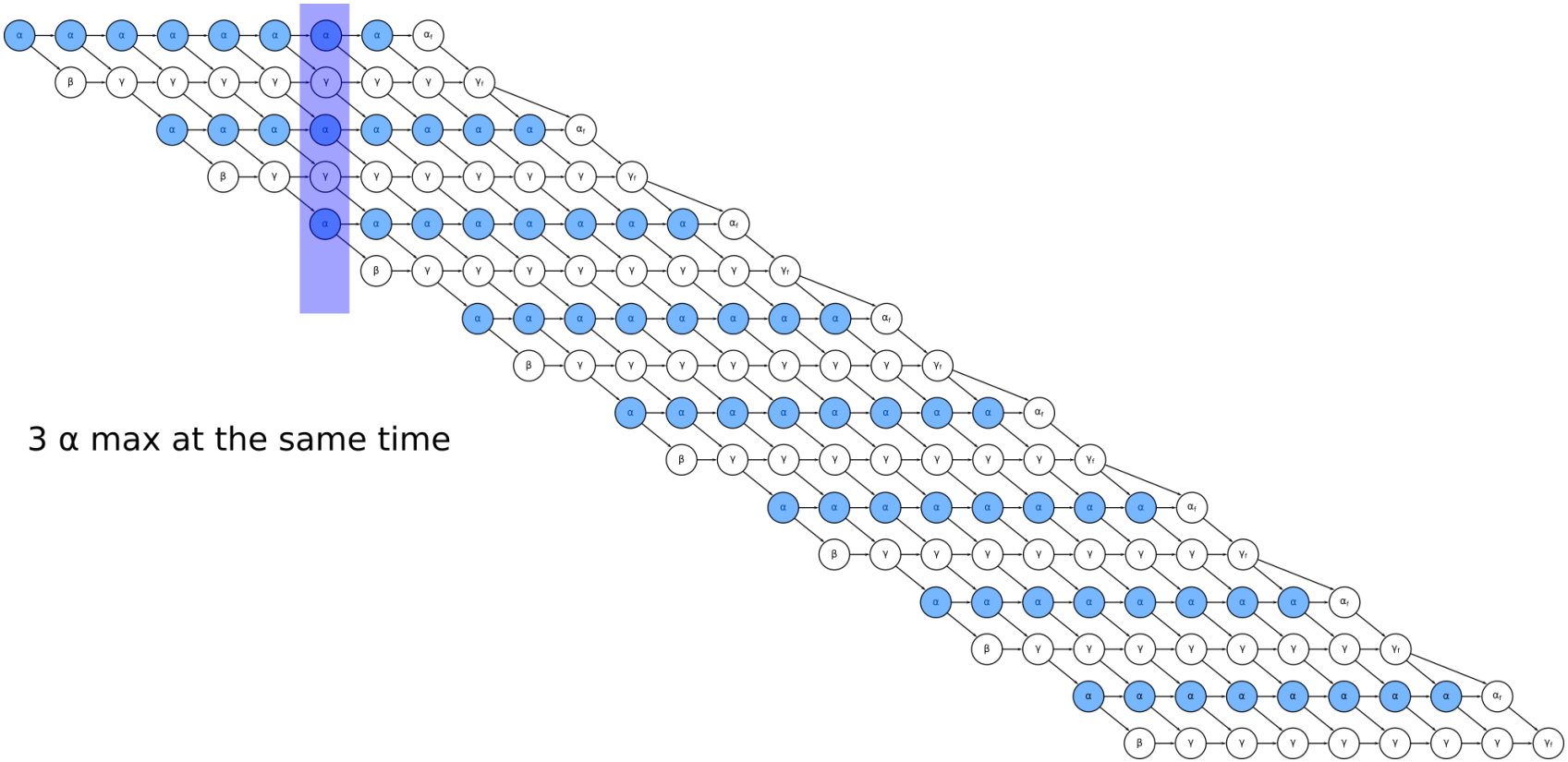
Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0;
2  for i ← 0 to s − 1 do
3      C ← 0;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C                    α
6          T[j] ← S
7      (C, S) ← T[s] + C                                      αf
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p′ mod 2^w                                  β
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do
13         (C, S) ← T[j] + m · p[j] + C                       γ
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                                      γf
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T;
```

S=8, All

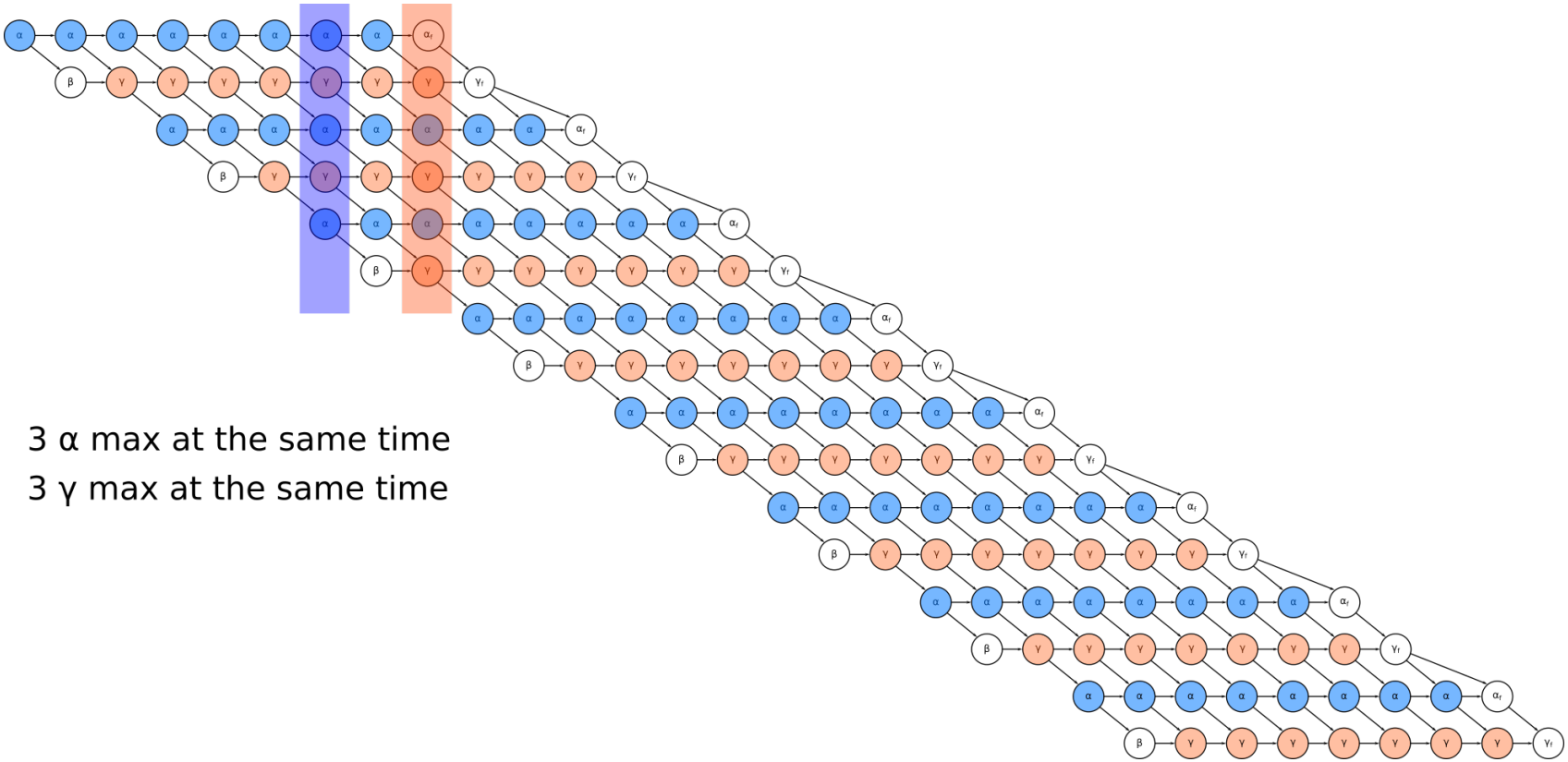


Alpha



3 α max at the same time

Gamma



3 α max at the same time

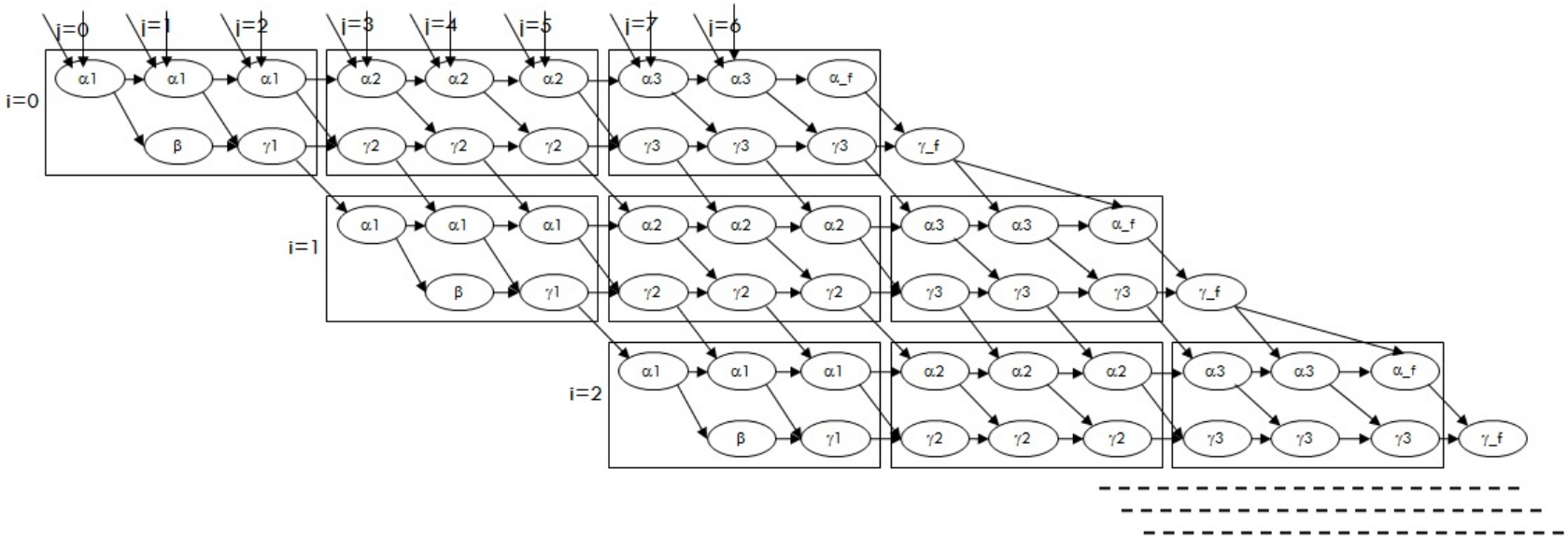
3 γ max at the same time

Resources

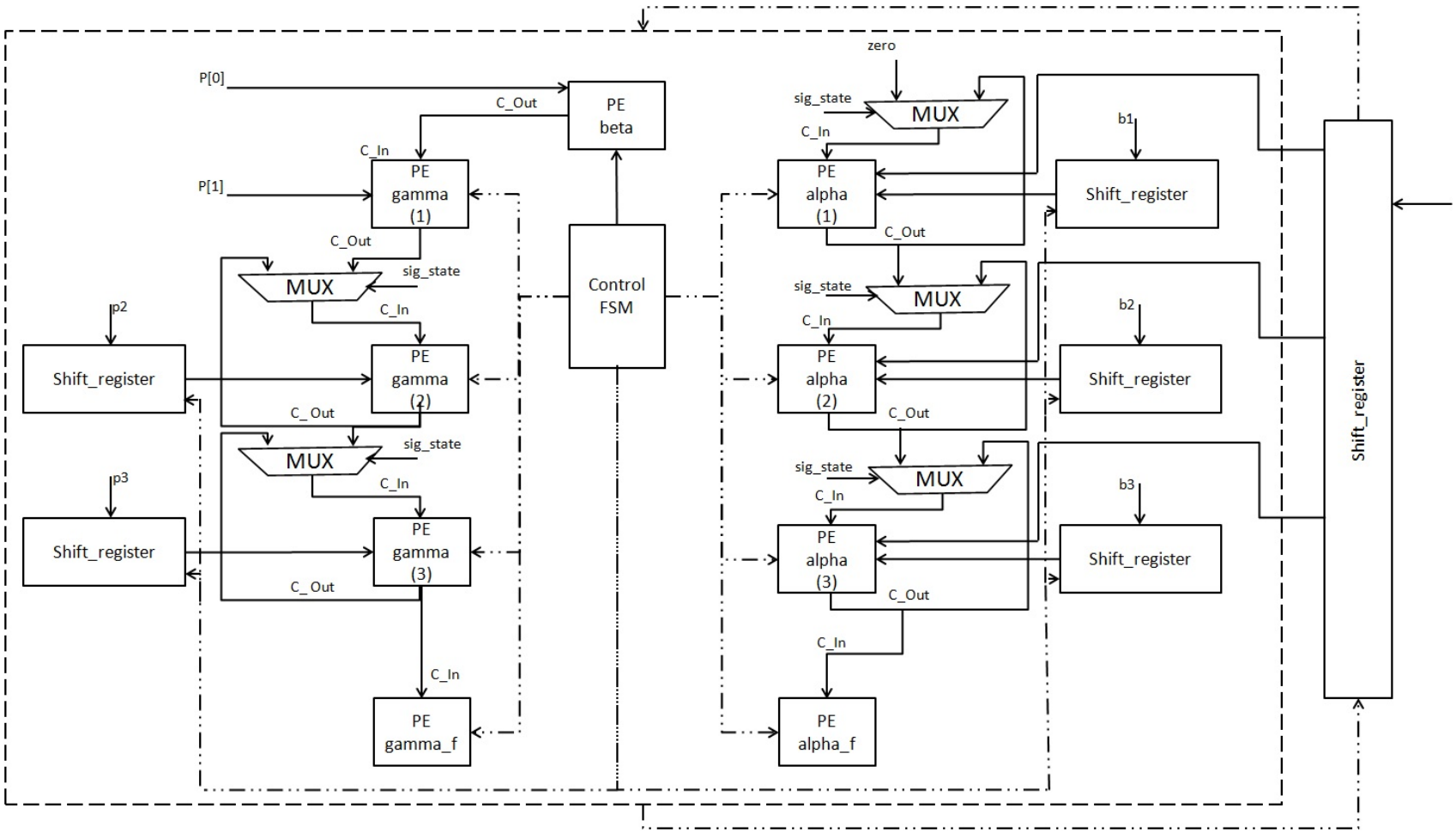
Our architecture requires:

- 3 α,
- 3 γ,
- 1 β,
- 1 α_f,
- 1 γ_f.

Regrouping



Block diagram



MMM architecture variants

CIOS (bits per word)	s=8	s=16	s=32	s=64
K=256	32	16	8	4
K=512	64	32	16	8
K=1024	128	64	32	16
K=2048	256	128	64	32
Clock cycles= $3 \times (s + nb)$	33	66	132	264
Number of cells	6 +3	12 +3	24 +3	48 +3

ECC results (Artix-7)

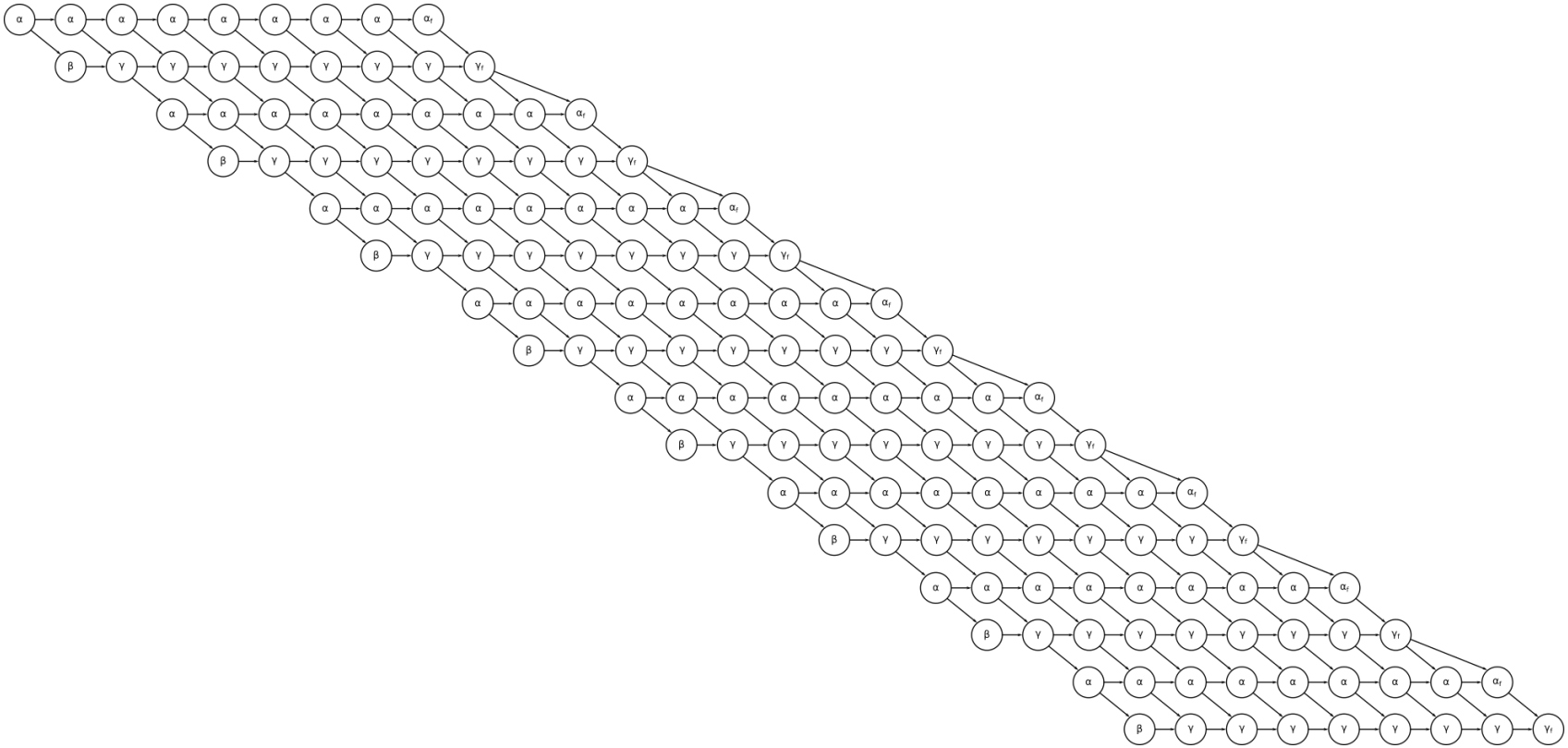
	Slice	DSPs	BRAM	Freq	Slice FF	Slice LUT
NW-8 (256)	3745	33	12	98	8281	9722
NW-16 (256)	3770	34	12	130	8313	9255
NW-8 (512)	7066	92	23	59	16500	20394
NW-16 (512)	7116	60	23	74	16501	19199

Conclusion

- Very efficient Montgomery Modular Multiplication with low latency.
- Give mixed results for a straightforward ECC implementation.
- Yet improvements are still possible: we should not wait the complete ending of an MMM to start the next.
- Should be particularly interesting for latency and throughput.

Thank you!

Any questions?



ECC results

... *et al.*	Curve	Device	Lut	Reg	Size (DSP)	Freq.
Bajard 2014	256 any	Kintex-7	4250	3532	1630 slices (46)	281
Bajard 2014	521 any	Kintex-7	7067	5882	2565 slices (91)	266
Bajard classic	256 any	–	7482	4605	– slices (46)	–
Guillermin	256 any	Stratix-2	–	–	9177 ALM (96)	157
Guillermin	512 any	Stratix-2	–	–	17017 ALM (244)	145
Güneysu	256 NIST	Virtex-4	–	–	1715 slices (32)	490
Yuan Ma	256 any	Virtex-4	5740	4876	4655 slices (37)	250
Yuan Ma	256 any	Virtex-5	4177	4792	1725 slices (37)	291
McIvor	256 any	Virtex-II	–	–	15755 slice	39
Us NW-8	256 any	Artix-7	9722	8281	3745 slices (33)	98
Us NW-8	512 any	Artix-7	20394	16500	7066 slices (92)	59
Us NW-16	256 any	Artix-7	9255	8313	3770 slices (34)	130
Us NW-16	512 any	Artix-7	19199	16501	7116 slices (60)	74