Efficient Algorithms for Large Prime Characteristic Fields And Their Application to Bilinear Pairings

Patrick Longa

MSR Security and Cryptography

CHES 2023

Prague, September 10-14

Microsoft Research

- Fix $a_i, b_i \in \mathbb{F}_p$ for $0 \leq i < t$, for a large prime p.

Suppose we need to compute:

$$\sum_{i=0}^{t-1} \pm a_i \times b_i \pmod{p}$$

- Fix $a_i, b_i \in \mathbb{F}_p$ for $0 \leq i < t$, for a large prime p.

Suppose we need to compute:

$$\sum_{i=0}^{t-1} \pm a_i \times b_i \pmod{p}$$

- This is a fundamental operation in several cryptologic applications.

The naïve way

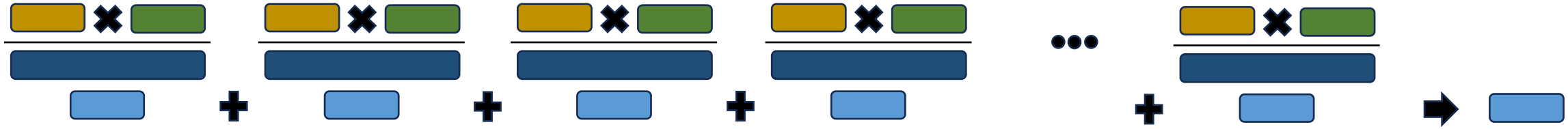
$$\sum_{i=0}^{t-1} \pm a_i \times b_i \pmod{p}$$

The naïve way

$$\sum_{i=0}^{t-1} \pm a_i \times b_i \pmod{p}$$

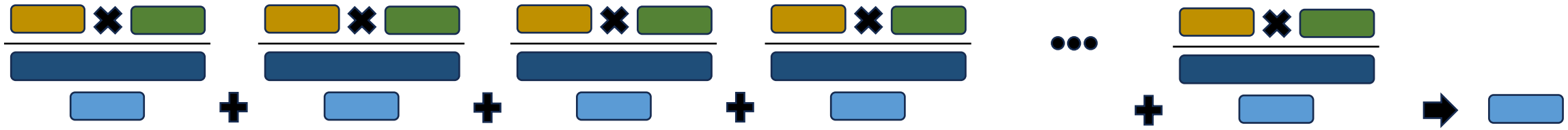
The naïve way

$$\sum_{i=0}^{t-1} \pm a_i \times b_i \pmod{p}$$



The naïve way

$$\sum_{i=0}^{t-1} \pm a_i \times b_i \pmod{p}$$



Cost: t int multiplications and t reductions.

- Assuming p is represented in n limbs:
  - Using Montgomery arithmetic, the cost is $t(2n^2 + n)$ digit-size multiplies.

The good-old lazy-reduction way

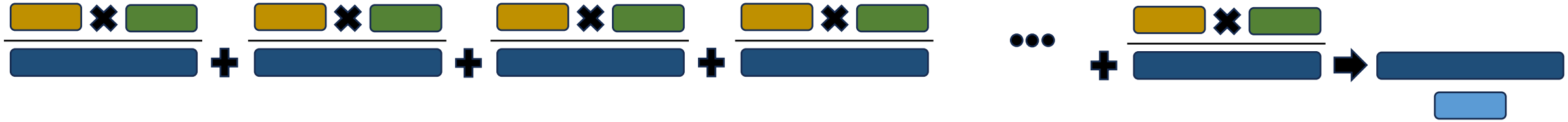
$$\left(\sum_{i=0}^{t-1} \pm a_i \times b_i\right) (\mathrm{mod}\ p)$$

The good-old lazy-reduction way

$$\left(\sum_{i=0}^{t-1} \pm a_i \times b_i\right) (\mathrm{mod}\ p)$$

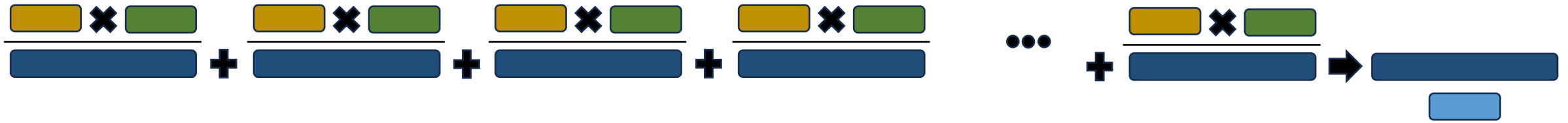
The good-old lazy-reduction way

$$\left(\sum_{i=0}^{t-1} \pm a_i \times b_i\right) (\mathrm{mod}\ p)$$



The good-old lazy-reduction way

$$\left(\sum_{i=0}^{t-1} \pm a_i \times b_i\right) \pmod{p}$$



Cost: t int multiplications and 1 reduction.

Trade-off: use of "double-precision" additions.

- Assuming p is represented in n limbs:
  - Using Montgomery arithmetic, it reduces the cost from $t(2n^2 + n)$ to $tn^2 + (n^2 + n)$ digit-size multiplies.

The good-old lazy-reduction way

Lazy reduction has been the de-facto technique.

The good-old lazy-reduction way

Lazy reduction has been the de-facto technique.

This includes:

- SW implementations of pairings, elliptic curves, dlog schemes, etc., for constructive and cryptanalytic purposes
  - In many cases in combination with (sub-quadratic) multiplication algorithms like Karatsuba.

The good-old lazy-reduction way

Lazy reduction has been the de-facto technique.

This includes:

- SW implementations of pairings, elliptic curves, dlog schemes, etc., for constructive and cryptanalytic purposes
  - In many cases in combination with (sub-quadratic) multiplication algorithms like Karatsuba.
- Also many HW implementations
  - Compact implementations sometimes prefer the naïve way.

A simple example: $\mathbb{F}_{p^2}$ multiplication

Constructing degree-2 extension field $\mathbb{F}_{p^2}$ of a finite field $\mathbb{F}_p$:

Fix $\mathbb{F}_{p^2} = \mathbb{F}_p(\alpha)$, with degree-2 irreducible polynomial $f(x)$ in $\mathbb{F}_p[x]$ s.t. $f(\alpha) = 0$

Let's assume $p \equiv 3 \bmod 4$, and take $\mathbb{F}_{p^2} = \mathbb{F}_p(i)/(i^2 + 1)$

A simple example: $\mathbb{F}_{p^2}$ multiplication

Constructing degree-2 extension field $\mathbb{F}_{p^2}$ of a finite field $\mathbb{F}_p$:

Fix $\mathbb{F}_{p^2} = \mathbb{F}_p(\alpha)$, with degree-2 irreducible polynomial $f(x)$ in $\mathbb{F}_p[x]$ s.t. $f(\alpha) = 0$

Let's assume $p \equiv 3 \bmod 4$, and take $\mathbb{F}_{p^2} = \mathbb{F}_p(i)/(i^2 + 1)$

Then,

$$a \times b = (a_0 + a_1 i) \times (b_0 + b_1 i) = \begin{cases} c_0 = (a_0 b_0 - a_1 b_1) (\bmod p) \\ c_1 = (a_0 b_1 + a_1 b_0) (\bmod p) \end{cases}$$

Cost: 4 int multiplications and 2 reductions.

A simple example: $\mathbb{F}_{p^2}$ multiplication

Since ~2010, (in SW) it's all Karatsuba + lazy rdc :

$$a \times b = \begin{cases} c_0 = (a_0 + a_1) \times (b_0 + b_1) - a_0 b_1 - a_1 b_0 \\ c_1 = (a_0 b_1 + a_1 b_0) \end{cases}$$

A simple example: $\mathbb{F}_{p^2}$ multiplication

Since ~2010, (in SW) it's all Karatsuba + lazy rdc :

$$a \times b = \begin{cases} c_0 = \overbrace{(a_0 + a_1) \times (b_0 + b_1)}^{\text{imul}} - \underbrace{a_0 b_1}_{\text{imul}} - \underbrace{a_1 b_0}_{\text{imul}} \longrightarrow \text{rdc} \\ c_1 = (a_0 b_1 + a_1 b_0) \longrightarrow \text{rdc} \end{cases}$$

Cost: 3 int multiplications and 2 reductions.

State-of-the-art: separated int multiplication and (Montgomery) reduction.

A new, more efficient way:
Generalizing interleaved modular multiplication

Do it the Montgomery way

- Let $a_i, b_i \in \mathbb{F}_p$ for $0 \leq i < t$, for a large prime p.
- Let $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}, n = \lceil l/w \rceil, l = \lceil \log p \rceil$ and w is the computer wordsize.
- Let $p' = -p^{-1} \log r$ for a certain radix-r.

Do it the Montgomery way

- Let $a_i, b_i \in \mathbb{F}_p$ for $0 \leq i < t$, for a large prime p.
- Let $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}, n = \lceil l/w \rceil, l = \lceil \log p \rceil$ and w is the computer wordsize.
- Let $p' = -p^{-1} \log r$ for a certain radix-r.

Using interleaved radix-r, we initialize $c = 0$ and execute:

$$c = (c + \sum_{i=0}^{t-1} a_{i,j} \times b_i + ((c + \sum_{i=0}^{t-1} a_{i,j} \times b_i)\, p' \bmod r)\, p)/r$$

from $j = 0$ to $\lceil l/\log r \rceil - 1$.

Do it the Montgomery way

- Let $a_i, b_i \in \mathbb{F}_p$ for $0 \leq i < t$, for a large prime p.
- Let $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}, n = \lceil l/w \rceil, l = \lceil \log p \rceil$ and w is the computer wordsize.
- Let $p' = -p^{-1} \log r$ for a certain radix-r.

Using interleaved radix-r, we initialize $c = 0$ and execute:

$$c = (c + \sum_{i=0}^{t-1} a_{i,j} \times b_i + ((c + \sum_{i=0}^{t-1} a_{i,j} \times b_i)\, p' \bmod r)\, p)/r$$

from $j = 0$ to $\lceil l/\log r \rceil - 1$.

SW example: coarsely integrated form (Simplified)

$u \leftarrow 0$

For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,j} \times b_i$$

$$q \leftarrow u \times p' \bmod r$$

$$u \leftarrow (u + q \times p)/r$$

$\vdots$

SW example: coarsely integrated form (Simplified)

$u \leftarrow 0$

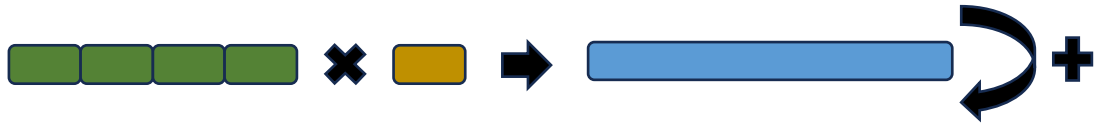
For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,j} \times b_i$$

$$q \leftarrow u \times p' \bmod r$$

$$u \leftarrow (u + q \times p)/r$$

$\vdots$



SW example: coarsely integrated form (Simplified)

$u \leftarrow 0$

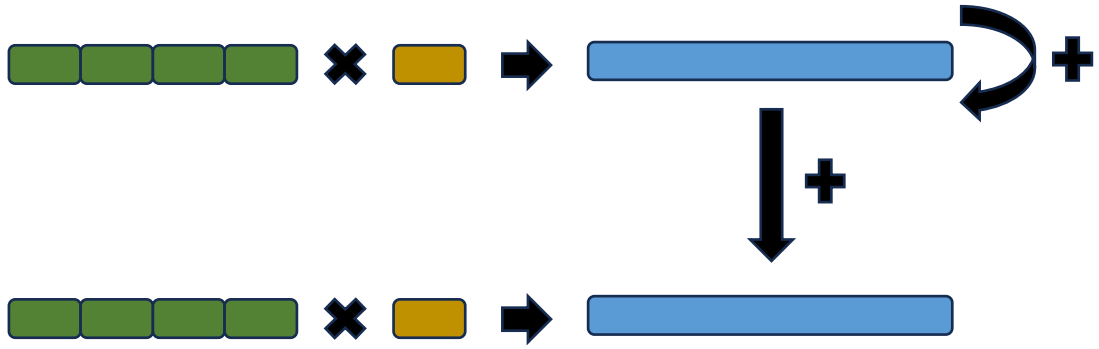
For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,j} \times b_i$$

$$q \leftarrow u \times p' \bmod r$$

$$u \leftarrow (u + q \times p)/r$$

$\vdots$



SW example: coarsely integrated form (Simplified)

$u \leftarrow 0$

For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,j} \times b_i$$

$$q \leftarrow u \times p' \bmod r$$

$$u \leftarrow (u + q \times p)/r$$

$\vdots$

Code example for pairing curve BLS12-381 ($\mathbb{F}_{p^2}$ mul, $t = 2$)

```
mov    rdx, [rcx+8]
MULADD64x384 [reg_p1+48], r9, r10, r11, r12, r13, r14, r8
mov    rdx, [rcx+56]
MULADD64x384 [reg_p1], r9, r10, r11, r12, r13, r14, r8
mov    rdx, [rip+u0]
mulx   rbp, rdx, r9
MULADD64x384 [rip+p0], r9, r10, r11, r12, r13, r14, r8
```

SW example: coarsely integrated form (Simplified)

$u \leftarrow 0$

For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,j} \times b_i$$

$$q \leftarrow u \times p' \bmod r$$

$$u \leftarrow (u + q \times p)/r$$

$\vdots$

Code example for pairing curve BLS12-381 ($\mathbb{F}_{p^2}$ mul, $t = 2$)

```
mov    rdx, [rcx+8]
MULADD64x384 [reg_p1+48], r9, r10, r11, r12, r13, r14, r8
mov    rdx, [rcx+56]
MULADD64x384 [reg_p1], r9, r10, r11, r12, r13, r14, r8
mov    rdx, [rip+u0]
mulx   rbp, rdx, r9
MULADD64x384 [rip+p0], r9, r10, r11, r12, r13, r14, r8
```

```
.macro MULADD64x384 M1, Z0, Z1, Z2, Z3, Z4, Z5, Z6, T0, T1
   mulx  \T0, \T1, \M1
   adox  \Z0, \T1;   adox  \Z1, \T0
   mulx  \T0, \T1, 8\M1
   adcx  \Z1, \T1;   adox  \Z2, \T0
   mulx  \T0, \T1, 16\M1
   adcx  \Z2, \T1;   adox  \Z3, \T0
   mulx  \T0, \T1, 24\M1
   adcx  \Z3, \T1;   adox  \Z4, \T0
   mulx  \T0, \T1, 32\M1
   adcx  \Z4, \T1;   adox  \Z5, \T0
   mulx  \T0, \T1, 40\M1
   adcx  \Z5, \T1;   adox  \Z6, \T0;   adcx  \Z6, rax
.endm
```

HW example: finely integrated form (Simplified)

$u \leftarrow 0$

For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,0} \times b_{i,j}$

$q \leftarrow u \times p' \bmod r$

$u \leftarrow (u + q \times p_0)/r$

For $k = 1$ to $\lceil l/\log r \rceil - 1$:

$u \leftarrow u + r^{k-1} \sum_{i=0}^{t-1} a_{i,k} \times b_{i,j}$

$u \leftarrow u + r^{k-1}(q \times p_k)$

$\vdots$

HW example: finely integrated form (Simplified)

$u \leftarrow 0$

For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,0} \times b_{i,j}$$

$q \leftarrow u \times p' \bmod r$

$u \leftarrow (u + q \times p_0)/r$

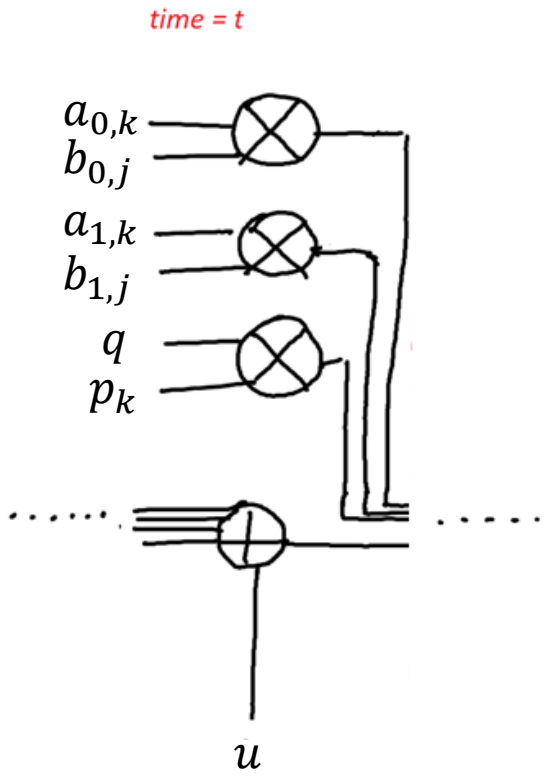
For $k = 1$ to $\lceil l/\log r \rceil - 1$:

$u \leftarrow u + r^{k-1} \sum_{i=0}^{t-1} a_{i,k} \times b_{i,j}$

$u \leftarrow u + r^{k-1}(q \times p_k)$

$\vdots$

A pipelined one-cycle-per-iteration architecture ($t = 2$)



HW example: finely integrated form (Simplified)

$u \leftarrow 0$

For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,0} \times b_{i,j}$$

$q \leftarrow u \times p' \bmod r$

$u \leftarrow (u + q \times p_0)/r$

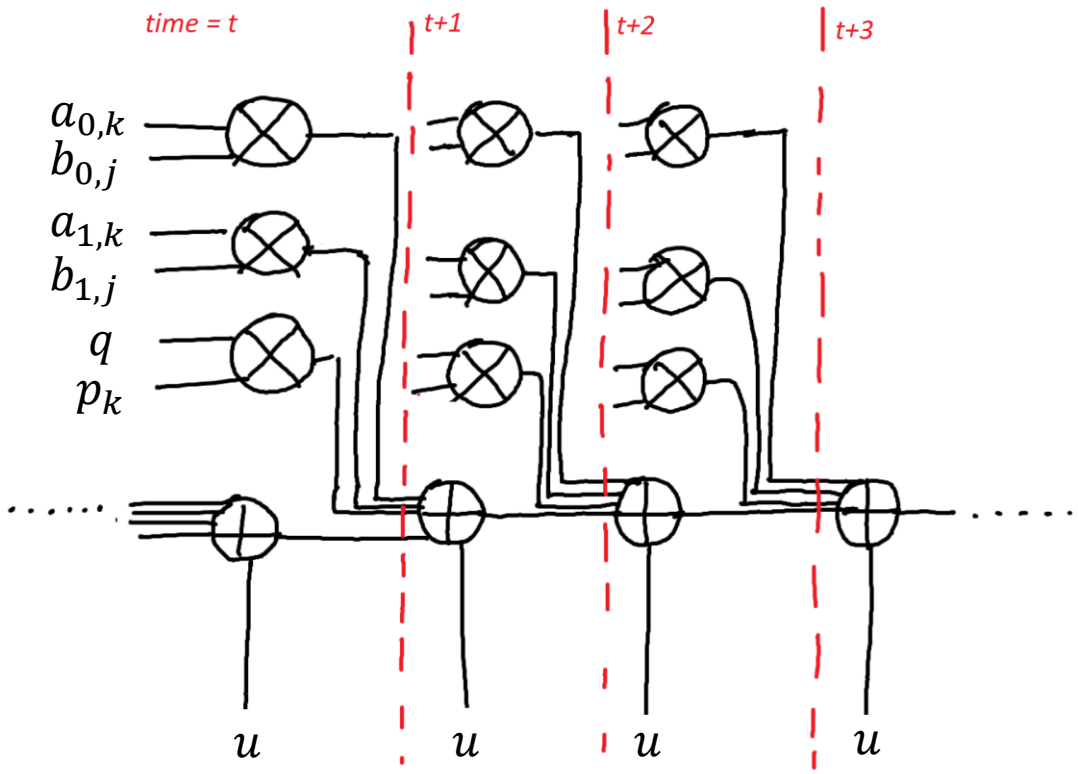
For $k = 1$ to $\lceil l/\log r \rceil - 1$:

$u \leftarrow u + r^{k-1} \sum_{i=0}^{t-1} a_{i,k} \times b_{i,j}$

$u \leftarrow u + r^{k-1}(q \times p_k)$

$\vdots$

A pipelined one-cycle-per-iteration architecture ($t = 2$)



HW example: finely integrated form (Simplified)

$u \leftarrow 0$

For $j = 0$ to $\lceil l/\log r \rceil - 1$:

$u \leftarrow u + \sum_{i=0}^{t-1} a_{i,0} \times b_{i,j}$

$q \leftarrow u \times p' \bmod r$

$u \leftarrow (u + q \times p_0)/r$

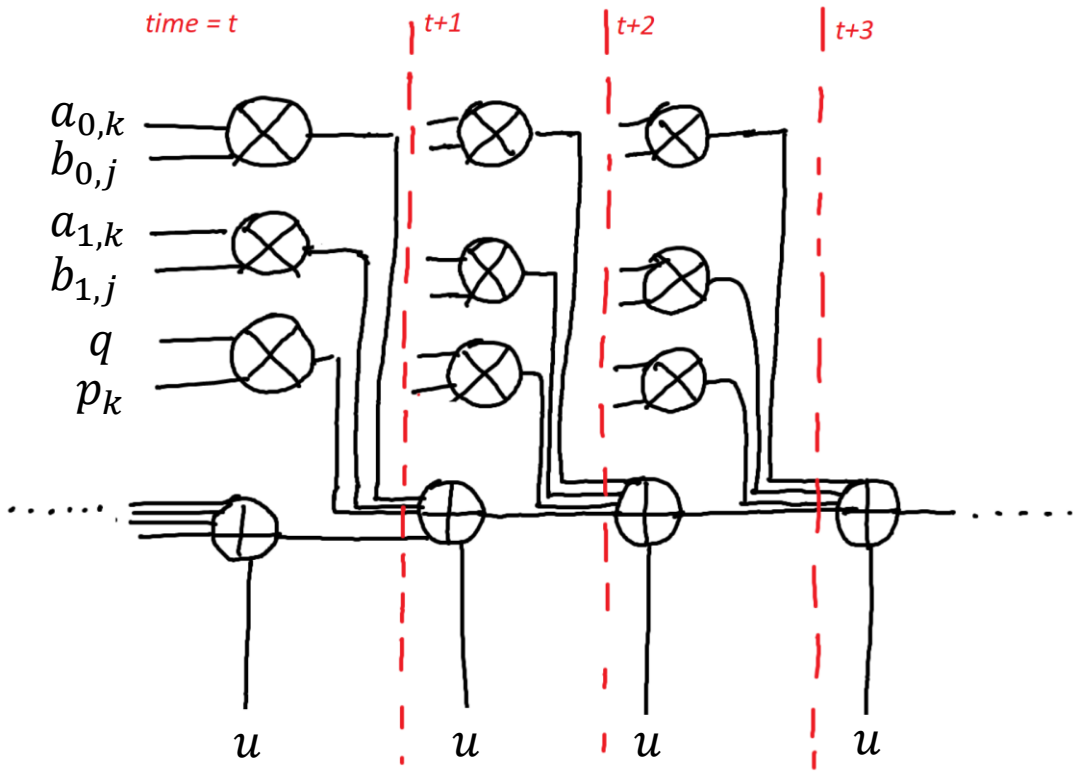
For $k = 1$ to $\lceil l/\log r \rceil - 1$:

$u \leftarrow u + r^{k-1} \sum_{i=0}^{t-1} a_{i,k} \times b_{i,j}$

$u \leftarrow u + r^{k-1}(q \times p_k)$

$\vdots$

A pipelined one-cycle-per-iteration architecture ($t = 2$)

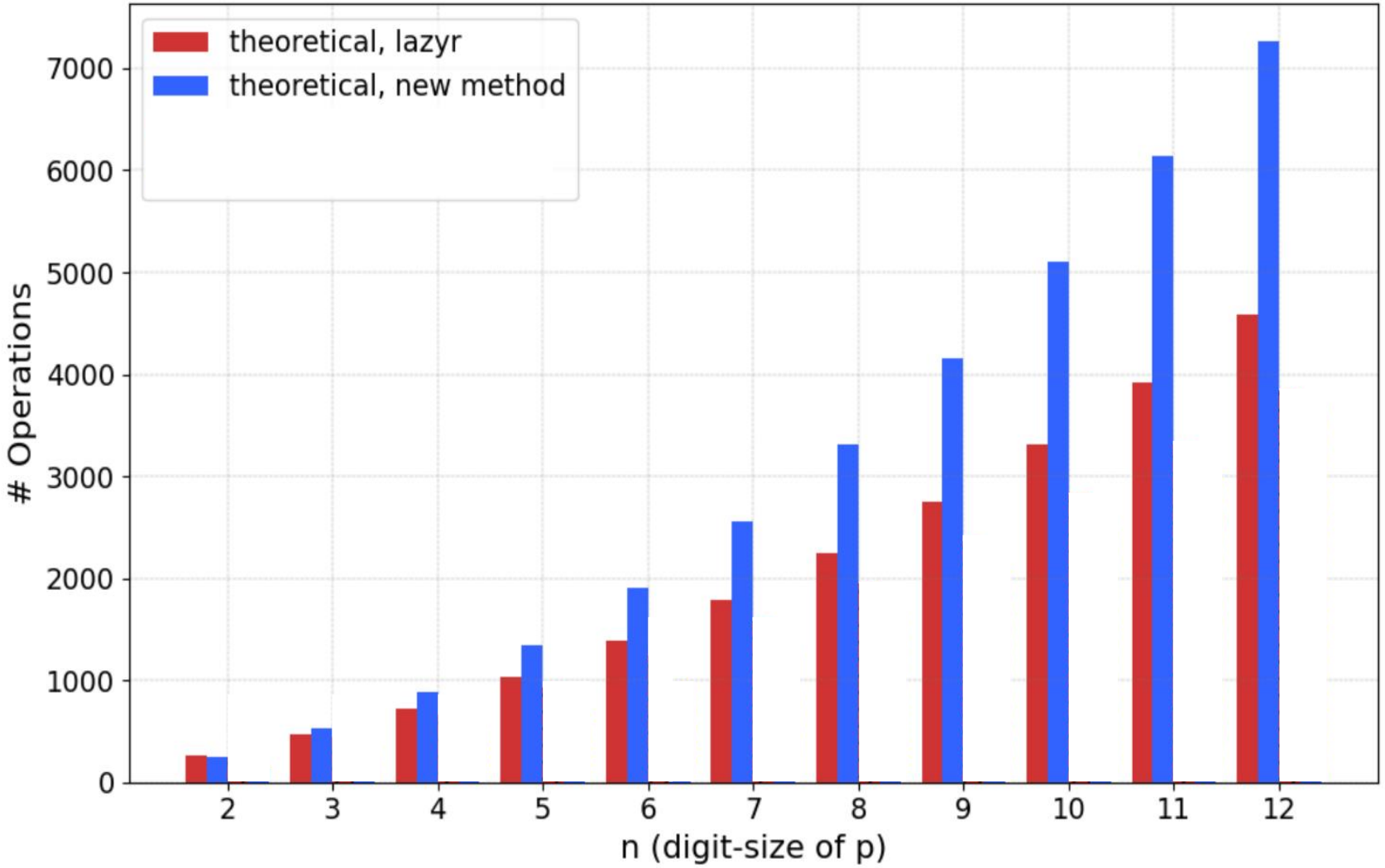


- Two instances optimal: 6 multipliers to execute one $\mathbb{F}_{p^2}$ mul in parallel.

Performance & (two) applications

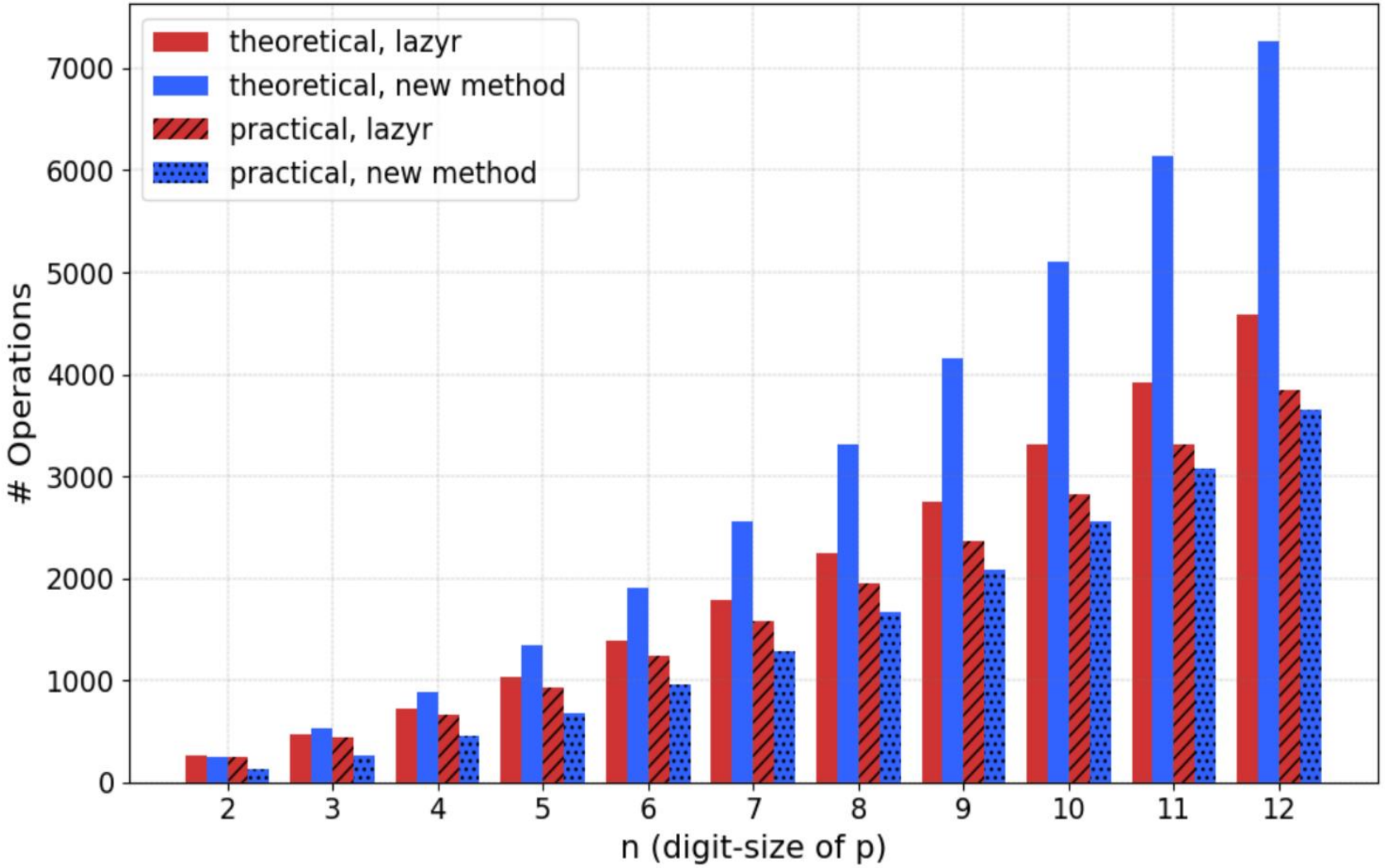
Lazy reduction vs. new method

Instruction count for one multiplication over $\mathbb{F}_{p^2}$ (x64 CPU)



Lazy reduction vs. new method

Instruction count for one multiplication over $\mathbb{F}_{p^2}$ (x64 CPU)



Bilinear pairings

Bilinear pairings

- The efficient implementation of arithmetic over $\mathbb{F}_{p^k}$ is critical for performance.
- The standard approach is to use a towering scheme.

Bilinear pairings

- The efficient implementation of arithmetic over $\mathbb{F}_{p^k}$ is critical for performance.
- The standard approach is to use a towering scheme.

- For example:

$\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta)$, with β a non-square

$\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[s]/(s^2 - \xi)$, with $\xi = \alpha + i$ a non-square

$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$, with $\xi = \alpha + i$ a non-cube

$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$ or $\mathbb{F}_{p^4}[t]/(t^3 - s)$ or $\mathbb{F}_{p^2}[w]/(w^6 - \xi)$ with $\xi = \alpha + i$ a non-square, non-cube.

Bilinear pairings

- Let $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$, with $\xi = \alpha + i$ non-cube.
- Let $a = a_0 + a_1 v + a_2 v^2 \in \mathbb{F}_{p^6}$, where $a_j = a_{j,0} + a_{j,1} i \in \mathbb{F}_{p^2}$.
- The multiplication $c = (c_0, c_1, c_2) = a \cdot b$ in $\mathbb{F}_{p^6}$ can be done as:

Bilinear pairings

- Let $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$, with $\xi = \alpha + i$ non-cube.
- Let $a = a_0 + a_1 v + a_2 v^2 \in \mathbb{F}_{p^6}$, where $a_j = a_{j,0} + a_{j,1} i \in \mathbb{F}_{p^2}$.
- The multiplication $c = (c_0, c_1, c_2) = a \cdot b$ in $\mathbb{F}_{p^6}$ can be done as:

$$
\begin{aligned}
c_{0,0} &= a_{0,0}b_{0,0} - a_{0,1}b_{0,1} + a_{1,0}(\alpha b_{2,0} - b_{2,1}) - a_{1,1}(b_{2,0} + \alpha b_{2,1}) + a_{2,0}(\alpha b_{1,0} - b_{1,1}) - a_{2,1}(b_{1,0} + \alpha b_{1,1}).\\
c_{0,1} &= a_{0,0}b_{0,1} + a_{0,1}b_{0,0} + a_{1,0}(b_{2,0} + \alpha b_{2,1}) + a_{1,1}(\alpha b_{2,0} - b_{2,1}) + a_{2,0}(b_{1,0} + \alpha b_{1,1}) + a_{2,1}(\alpha b_{1,0} - b_{1,1}).\\
c_{1,0} &= a_{0,0}b_{1,0} - a_{0,1}b_{1,1} + a_{1,0}b_{0,0} - a_{1,1}b_{0,1} + a_{2,0}(\alpha b_{2,0} - b_{2,1}) - a_{2,1}(\alpha b_{2,1} + b_{2,0}).\\
c_{1,1} &= a_{0,0}b_{1,1} + a_{0,1}b_{1,0} + a_{1,0}b_{0,1} + a_{1,1}b_{0,0} + a_{2,0}(\alpha b_{2,1} + b_{2,0}) + a_{2,1}(\alpha b_{2,0} - b_{2,1}).\\
c_{2,0} &= a_{0,0}b_{2,0} - a_{0,1}b_{2,1} + a_{1,0}b_{1,0} - a_{1,1}b_{1,1} + a_{2,0}b_{0,0} - a_{2,1}b_{0,1}.\\
c_{2,1} &= a_{0,0}b_{2,1} + a_{0,1}b_{2,0} + a_{1,0}b_{1,1} + a_{1,1}b_{1,0} + a_{2,0}b_{0,1} + a_{2,1}b_{0,0}.
\end{aligned}
$$

— pre-computed

Performance on x64 CPU (curve BLS12-381)

Reference	Strategy	Speed		Memory	
		cc	%	bytes	%
	$\mathbb{F}_{p^2}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	566	-	1,920	-
This work	Interleaved over $\mathbb{F}_{p^2}$	**355**	**-37%**	**1,104**	**-43%**

(*) Obtained on a 3.4GHz Intel Core i7-6700 (Skylake) processor.

Performance on x64 CPU (curve BLS12-381)

Reference	Strategy	Speed		Memory	
		cc	%	bytes	%
	$\mathbb{F}_{p^2}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	566	-	1,920	-
This work	Interleaved over $\mathbb{F}_{p^2}$	**355**	**-37%**	**1,104**	**-43%**
	$\mathbb{F}_{p^6}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	3,376	-	6,320	-
This work	New method over $\mathbb{F}_{p^2}$. Karat over $\mathbb{F}_{p^6}$	2,695	-20%	2,416	-62%
This work	New method over $\mathbb{F}_{p^6}$	**2,342**	**-31%**	**2,104**	**-67%**

(*) Obtained on a 3.4GHz Intel Core i7-6700 (Skylake) processor.

Performance on x64 CPU (curve BLS12-381)

Reference	Strategy	Speed		Memory	
		cc	%	bytes	%
	$\mathbb{F}_{p^2}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	566	-	1,920	-
This work	Interleaved over $\mathbb{F}_{p^2}$	**355**	**-37%**	**1,104**	**-43%**
	$\mathbb{F}_{p^6}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	3,376	-	6,320	-
This work	New method over $\mathbb{F}_{p^2}$. Karat over $\mathbb{F}_{p^6}$	2,695	-20%	2,416	-62%
This work	New method over $\mathbb{F}_{p^6}$	**2,342**	**-31%**	**2,104**	**-67%**
	$\mathbb{F}_{p^{12}}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	10,061	-	16,040	-
This work	New method over $\mathbb{F}_{p^6}$. Karat over $\mathbb{F}_{p^{12}}$	**7,858**	**-22%**	3,928	-76%
This work	New method over $\mathbb{F}_{p^{12}}$	8,315	-17%	**3,544**	**-78%**

(*) Obtained on a 3.4GHz Intel Core i7-6700 (Skylake) processor.

Performance on x64 CPU (curve BLS12-381)

Reference	Strategy	Speed		Memory	
		cc	%	bytes	%
	$\mathbb{F}_{p^2}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	566	-	1,920	-
This work	Interleaved over $\mathbb{F}_{p^2}$	**355**	**-37%**	**1,104**	**-43%**
	$\mathbb{F}_{p^6}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	3,376	-	6,320	-
This work	New method over $\mathbb{F}_{p^2}$. Karat over $\mathbb{F}_{p^6}$	2,695	-20%	2,416	-62%
This work	New method over $\mathbb{F}_{p^6}$	**2,342**	**-31%**	**2,104**	**-67%**
	$\mathbb{F}_{p^{12}}$ mul				
RELIC	Separated mul/rdc. Karat+lazyr	10,061	-	16,040	-
This work	New method over $\mathbb{F}_{p^6}$. Karat over $\mathbb{F}_{p^{12}}$	**7,858**	**-22%**	3,928	-76%
This work	New method over $\mathbb{F}_{p^{12}}$	8,315	-17%	**3,544**	**-78%**
	Full pairing				
RELIC	Separated mul/rdc. Karat+lazyr	3.15×10^6	-	23,198	-
This work	New method over $\mathbb{F}_{p^6}$. Karat over $\mathbb{F}_{p^{12}}$	$\mathbf{2.29 \times 10^6}$	**-27%**	12,752	-45%
This work	New method over $\mathbb{F}_{p^{12}}$	2.30×10^6	-27%	**11,792**	**-49%**

(*) Obtained on a 3.4GHz Intel Core i7-6700 (Skylake) processor.

Supersingular isogeny-based schemes

Supersingular isogeny-based schemes

Efficient Algorithms for Large Prime Characteristic Fields and Their Application to Bilinear Pairings and Supersingular Isogeny-Based Protocols

Abstract. We propose a novel approach that generalizes interleaved modular multiplication algorithms to the computation of sums of products over large prime fields. This operation has widespread use and is at the core of many cryptographic applications. The method reformulates the widely used lazy reduction technique, crucially avoiding the need for storage and computation of "double-precision" operations. Moreover, it can be easily adapted to the different methods that exist to compute modular multiplication, producing algorithms that are significantly more efficient and memory-friendly. We showcase the performance of the proposed approach in the computation of multiplication over an extension field $\mathbb{F}_{p^k}$, and demonstrate its impact in two popular cryptographic settings: bilinear pairings and supersingular isogeny-based protocols. For the former, we obtain a 1.37× speedup in the computation of a full optimal ate pairing over the popular BLS12-381 curve on an x64 Intel processor; and for the latter, we show a speedup of up to 1.30× in the computation of the SIKE protocol on the same Intel platform.

Keywords: Sum of products · prime fields · extension fields · bilinear pairings · BLS12-381 · supersingular isogeny-based cryptography · SIKE · efficient computation.

Supersingular isogeny-based schemes

Efficient Algorithms for Large Prime Characteristic Fields and Their Application to Bilinear Pairings ~~and Supersingular Isogeny-Based Protocols~~

Abstract. We propose a novel approach that generalizes interleaved modular multiplication algorithms to the computation of sums of products over large prime fields. This operation has widespread use and is at the core of many cryptographic applications. The method reformulates the widely used lazy reduction technique, crucially avoiding the need for storage and computation of "double-precision" operations. Moreover, it can be easily adapted to the different methods that exist to compute modular multiplication, producing algorithms that are significantly more efficient and memory-friendly. We showcase the performance of the proposed approach in the computation of multiplication over an extension field $\mathbb{F}_{p^k}$, and demonstrate its impact in two popular cryptographic settings: bilinear pairings ~~and supersingular isogeny-based protocols.~~ For the former, we obtain a 1.37× speedup in the computation of a full optimal ate pairing over the popular BLS12-381 curve on an x64 Intel processor; ~~and for the latter, we show a speedup of up to 1.30× in the computation of the SIKE protocol on the same Intel platform~~.

Keywords: Sum of products · prime fields · extension fields · bilinear pairings · BLS12-381 ~~· supersingular isogeny-based cryptography · SIKE~~ · efficient computation.

- July 2022: a series of papers starting with Castryck and Decru breaks SIDH/SIKE in polynomial time.

Supersingular isogeny-based schemes: the comeback

SQISign

Supersingular isogeny-based schemes: the comeback
SQISign

- **Main idea:** take a curve E_A as public key and $\mathrm{End}(E_A)$ as the private key. A SQISign signer needs to prove knowledge of $\mathrm{End}(E_A)$ by solving the isogeny problem.

Supersingular isogeny-based schemes: the comeback
SQISign

- **Main idea:** take a curve E_A as public key and $\mathrm{End}(E_A)$ as the private key. A SQISign signer needs to prove knowledge of $\mathrm{End}(E_A)$ by solving the isogeny problem.
- SQISign offers the smallest (pk + sign) sizes in the PQC world.

Primitive	NIST level	pk (bytes)	sign (bytes)
SQISign	1	**64**	**204**
Dilithium	1	1312	2420

Supersingular isogeny-based schemes: the comeback
SQISign

- **Challenge:** it’s very slow.

Supersingular isogeny-based schemes: the comeback
SQISign

- **Challenge:** it's very slow.
- Underlying arithmetic is constructed over $\mathbb{F}_{p^2}$!
- The proposed method speeds up sign + verify by $\sim 1.68\times$ (NIST level 1 using new 254-bit prime p_{3923})
  - E.g., verify (Skylake) cycles are reduced from 60 million to 22 million.

Supersingular isogeny-based schemes: the comeback
SQISign

- **Challenge:** it's very slow.
- Underlying arithmetic is constructed over $\mathbb{F}_{p^2}$!
- The proposed method speeds up sign + verify by $\sim 1.68\times$ (NIST level 1 using new 254-bit prime p_{3923})
  - E.g., verify (Skylake) cycles are reduced from 60 million to 22 million.

- SQISign is in Round 1 of NIST's Call for Additional Signatures Schemes.

https://sqisign.org/

Efficient Algorithms for Large Prime Characteristic Fields And Their Application to Bilinear Pairings

Patrick Longa

MSR Security and Cryptography

http://patricklonga.com
Twitter: @PatrickLonga

CHES 2023

Prague, September 10-14

Microsoft Research