Four$\mathbb{Q}$ on Embedded Devices with Strong Countermeasures Against Side-Channel Attacks

CHES 2017
September 26-28, Taipei, Taiwan

Zhe Liu Patrick Longa
Geovandro C. C. F. Pereira
Oscar Reparaz Hwajeong Seo

UNIVERSITY OF WATERLOO | IQC Institute for Quantum Computing

Zhe Liu Patrick Longa

Geovandro C. C. F. Pereira

Oscar Reparaz Hwajeong Seo

Context on modern elliptic curves

- 1996, P. Kocher initiates Simple Power Analysis (SPA) attacks (timing).
  - 1999, SPA evolves to Differential Power Analysis (DPA) and Template attacks.

Context on modern elliptic curves

- 1996, P. Kocher initiates Simple Power Analysis (SPA) attacks (timing).
  - 1999, SPA evolves to Differential Power Analysis (DPA) and Template attacks.
- **1999, FIPS 186-2 is published**
  - NIST publishes the 15 popular NIST (**Weierstrass**) curves along with ECDSA.

Context on modern elliptic curves

- 1996, P. Kocher initiates Simple Power Analysis (SPA) attacks (timing).
  - 1999, SPA evolves to Differential Power Analysis (DPA) and Template attacks.
- **1999, FIPS 186-2 is published**
  - NIST publishes the 15 popular NIST (**Weierstrass**) curves along with ECDSA.
- **New requirements imposed to ECC**
  - Constant-time algorithms
    - Complete formulas (achieved by models such as (Twisted) **Edwards** curves).
  - Provenance

Context on modern elliptic curves

- 1996, P. Kocher initiates Simple Power Analysis (SPA) attacks (timing).
  - 1999, SPA evolves to Differential Power Analysis (DPA) and Template attacks.
- **1999, FIPS 186-2 is published**
  - NIST publishes the 15 popular NIST (**Weierstrass**) curves along with ECDSA.
- **New requirements imposed to ECC**
  - Constant-time algorithms
    - Complete formulas (achieved by models such as (Twisted) **Edwards** curves).
  - Provenance
- **2015**, NIST holds a workshop for new ECC standardization.

Next-generation elliptic curves

Farrel-Moriarity-Melkinov-Paterson [NIST ECC Workshop 2015]:

*"… the real motivation for work in CFRG is the **better performance** and **side-channel resistance of new curves** developed by academic cryptographers over the last decade."*

State-of-the-art ECC: Four$\mathbb{Q}$
[Costello-Longa, ASIACRYPT 2015]

Speed (in thousands of cycles) to compute variable-base scalar multiplication on different computer classes.

Platform	Four$\mathbb{Q}$	Curve25519	Speedup ratio
Intel Haswell processor, **desktop class**	**56**	162	**2.9x**
ARM Cortex-A15, **smartphone class**	**132**	315	**2.4x**
ARM Cortex-M4, **microcontroller class**	**470**	907 / 1,424	**1.9 / 3.0x**

State-of-the-art ECC: Four$\mathbb{Q}$
[Costello-Longa, ASIACRYPT 2015]

$$E/\mathbb{F}_{p^2}: -x^2 + y^2 = 1 + dx^2y^2$$

$d = 125317048443780598345676279555970305165i + 4205857648805777768770,$

$p = 2^{127} - 1$, $i^2 = -1$, $\#E = 392 \cdot N$, where N is a 246-bit prime.

State-of-the-art ECC: Four$\mathbb{Q}$
(Costello-Longa, ASIACRYPT 2015)

$$E/\mathbb{F}_{p^2}: -x^2 + y^2 = 1 + dx^2y^2$$

$$d = 125317048443780598345676279555970305165i + 4205857648805777768770,$$

$p = 2^{127} - 1$, $i^2 = -1$, $\#E = 392 \cdot N$, where N is a 246-bit prime.

- Fastest (large char) ECC addition laws **are complete** on E
- E is equipped with *two* endomorphisms:
  - E is a degree-2 $\mathbb{Q}$-curve: endomorphism ψ
  - E has CM by order of $D = -40$: endomorphism ϕ

State-of-the-art ECC: Four$\mathbb{Q}$
(Costello-Longa, ASIACRYPT 2015)

$$E/\mathbb{F}_{p^2}: -x^2 + y^2 = 1 + dx^2y^2$$

$$d = 125317048443780598345676279555970305165i + 4205857648805777768770,$$

$p = 2^{127} - 1$, $i^2 = -1$, $\#E = 392 \cdot N$, where N is a 246-bit prime.

- Fastest (large char) ECC addition laws **are complete** on E
- E is equipped with *two* endomorphisms:
  - E is a degree-2 $\mathbb{Q}$-curve: endomorphism ψ
  - E has CM by order of $D = -40$: endomorphism ϕ
- $\psi(P) = [\lambda_\psi]P$ and $\phi(P) = [\lambda_\phi]P$ for all $P \in E[N]$ and $m \in [0, 2^{256})$

$$m \mapsto (a_1, a_2, a_3, a_4)$$

$$[m]P = [a_1]P + [a_2]\phi(P) + [a_3]\psi(P) + [a_4]\psi(\phi(P))$$

Optimal 4-Way Scalar Decompositions

$$m \mapsto (a_1, a_2, a_3, a_4)$$

Proposition: for all $m \in [0, 2^{256}\rangle$, decomposition yields four $a_i \in [0, 2^{64}\rangle$ with a_1 odd.

$$m = 42453556751700041597675664513313229052985088397396902723728803518727612539248$$

$$\begin{array}{llll} a_1 & = 13045455764875651153 & & P \\ a_2 & = 9751504369311420685 & & \phi(P) \\ a_3 & = 5603607414148260372 & & \psi(P) \\ a_4 & = 8360175734463666813 & & \psi(\phi(P)) \end{array}$$

Optimal 4-Way Scalar Decompositions

$$m \mapsto (a_1, a_2, a_3, a_4)$$

Proposition: for all $m \in [0, 2^{256}\rangle$, decomposition yields four $a_i \in [0, 2^{64}\rangle$ with a_1 odd.

$$m = 42453556751700041597675664513313229052985088397396902723728803518727612539248$$

$$\begin{array}{lll} a_1 & = 13045455764875651153 & P \\ a_2 & = 9751504369311420685 & \phi(P) \\ a_3 & = 5603607414148260372 & \psi(P) \\ a_4 & = 8360175734463666813 & \psi(\phi(P)) \end{array}$$

Multi-Scalar Recoding

Step 1: recode a_1 to signed non-zero representation

Step 2: recode a_2, a_3 and a_4 by “sign-aligning” columns

$$a_1 = 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1$$
$$a_2 = 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1$$
$$a_3 = 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0$$
$$a_4 = 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1$$

$$a_1 = 1, \bar{1}, 1, \bar{1}, 1, 1, \bar{1}, 1, \bar{1}, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, 1, \bar{1}, 1, 1, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, \bar{1}, 1, 1, 1, \bar{1}, \bar{1}, 1, 1, \bar{1}, \bar{1}, 1, 1, 1, 1, 1, 1, \bar{1}, \bar{1}, 1, 1, 1, 1, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, 1, \bar{1}, \bar{1}, \bar{1}$$
$$a_2 = 1, \bar{1}, 0, 0, 0, 1, 0, 0, \bar{1}, 1, 0, \bar{1}, \bar{1}, 0, 1, 0, 0, 0, 1, 1, \bar{1}, 0, \bar{1}, 1, 0, \bar{1}, 0, 0, 1, 0, \bar{1}, 1, 1, 0, \bar{1}, 1, 0, 0, 1, 1, 1, \bar{1}, \bar{1}, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, \bar{1}, \bar{1}, 0, 0, 1, \bar{1}, 0, 0, \bar{1}, \bar{1}$$
$$a_3 = 0, 0, 1, 0, 1, 0, \bar{1}, 1, 0, 0, \bar{1}, 0, 0, 0, 1, 0, 0, 0, 0, 1, \bar{1}, \bar{1}, \bar{1}, 0, \bar{1}, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, \bar{1}, 0, \bar{1}, 0, 0, 1, \bar{1}, 0, 0, 0, 1, \bar{1}, 1, \bar{1}, 0, 0$$
$$a_4 = 1, \bar{1}, 0, \bar{1}, 1, 1, \bar{1}, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, \bar{1}, 0, 0, 0, 0, \bar{1}, 0, 0, 1, \bar{1}, 0, 1, 0, \bar{1}, \bar{1}, 0, 1, 0, 0, 0, 1, \bar{1}, 0, 0, 0, 1, 1, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 0, \bar{1}, 1, 0, \bar{1}, \bar{1}, 0, 0, 0, 0, 0, \bar{1}, \bar{1}$$

Multi-Scalar Recoding

Step 1: recode a_1 to signed non-zero representation

Step 2: recode a_2, a_3 and a_4 by “sign-aligning” columns

$$a_1 = 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1$$
$$a_2 = 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1$$
$$a_3 = 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0$$
$$a_4 = 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1$$

$$a_1 = 1, \bar{1}, 1, \bar{1}, 1, 1, \bar{1}, 1, \bar{1}, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, 1, \bar{1}, 1, 1, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, \bar{1}, 1, 1, 1, \bar{1}, \bar{1}, 1, 1, \bar{1}, \bar{1}, 1, 1, 1, 1, 1, 1, \bar{1}, \bar{1}, 1, 1, 1, 1, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 1, \bar{1}, 1, \bar{1}, \bar{1}, \bar{1}$$
$$a_2 = 1, \bar{1}, 0, 0, 0, 1, 0, 0, \bar{1}, 1, 0, \bar{1}, \bar{1}, 0, 1, 0, 0, 0, 1, 1, \bar{1}, 0, \bar{1}, 1, 0, \bar{1}, 0, 0, 1, 0, \bar{1}, 1, 1, 0, \bar{1}, 1, 0, 0, 1, 1, 1, \bar{1}, \bar{1}, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, \bar{1}, \bar{1}, 0, 0, 1, \bar{1}, 0, 0, \bar{1}, \bar{1}$$
$$a_3 = 0, 0, 1, 0, 1, 0, \bar{1}, 1, 0, 0, \bar{1}, 0, 0, 0, 1, 0, 0, 0, 0, 1, \bar{1}, \bar{1}, \bar{1}, 0, \bar{1}, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, \bar{1}, 0, \bar{1}, 0, 0, 1, \bar{1}, 0, 0, 0, 1, \bar{1}, 1, \bar{1}, 0, 0$$
$$a_4 = 1, \bar{1}, 0, \bar{1}, 1, 1, \bar{1}, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, \bar{1}, 0, 0, 0, 0, \bar{1}, 0, 0, 1, \bar{1}, 0, 1, 0, \bar{1}, \bar{1}, 0, 1, 0, 0, 0, 1, \bar{1}, 0, 0, 0, 1, 1, 1, \bar{1}, \bar{1}, \bar{1}, \bar{1}, 0, \bar{1}, 1, 0, \bar{1}, \bar{1}, 0, 0, 0, 0, 0, \bar{1}, \bar{1}$$

column signs s_i $[+ - + - + + - + - + - - - - + - + - + + - - - + - - + + + - - + + - - + + + + + + - - + + + + + - - - - + - + - - - - + - + - - -]$

digits d_i $[6, 6, 3, 5, 7, 6, 7, 3, 2, 2, 3, 2, 2, 1, 8, 1, 5, 1, 6, 8, 8, 3, 4, 2, 3, 6, 3, 1, 6, 5, 2, 6, 4, 5, 6, 2, 5, 1, 4, 2, 8, 6, 2, 2, 2, 8, 7, 8, 5, 7, 5, 7, 2, 5, 8, 4, 6, 5, 1, 4, 4, 3, 3, 6, 6]$

Regular Multi-Scalar Multiplication

column signs s_i $[+ - + - + + - + - + - - - - + - + - + + - - - + - - + + + - - + + - - + + + + + + - - + + + + + - - - - + - + - - - - + - + - - -]$

digits d_i $[6, 6, 3, 5, 7, 6, 7, 3, 2, 2, 3, 2, 2, 1, 8, 1, 5, 1, 6, 8, 8, 3, 4, 2, 3, 6, 3, 1, 6, 5, 2, 6, 4, 5, 6, 2, 5, 1, 4, 2, 8, 6, 2, 2, 2, 8, 7, 8, 5, 7, 5, 7, 2, 5, 8, 4, 6, 5, 1, 4, 4, 3, 3, 6, 6]$

Execution

➔ *Load* $Q = T[6] = P + \phi(P) + \psi(\phi(P))$

64 times

➔ $Q = 2Q - T[6] = P + \phi(P) + \psi(\phi(P))$

➔ $Q = 2Q + T[3] = 3P + 2\phi(P) + \psi(P) + 2\psi(\phi(P))$

➔ $Q = 2Q - T[5] = 5P + 4\phi(P) + 2\psi(P) + 3\psi(\phi(P))$

⋮

T[1]	P
T[2]	$P + \phi(P)$
T[3]	$P + \psi(P)$
T[4]	$P + \phi(P) + \psi(P)$
T[5]	$P + \psi(\phi(P))$
T[6]	$P + \phi(P) + \psi(\phi(P))$
T[7]	$P + \psi(P) + \psi(\phi(P))$
T[8]	$P + \phi(P) + \psi(P) + \psi(\phi(P))$

- **Regular execution** (exactly 64 DBLS and 64 ADDs) facilitates protection against timing/SSCA attacks.
- Reduced number of precomputations (only 8 points).

Regular Multi-Scalar Multiplication

column signs s_i $\left[+ - + - + + - + - + - - - - + - + - + + - - - + - - + + + - - + + - - + + + + + + - - + + + + + - - - - + - + - - - - + - + - - - \right]$

digits d_i $\left[\cancel{6}, 6, 3, 5, 7, 6, 7, 3, 2, 2, 3, 2, 2, 1, 8, 1, 5, 1, 6, 8, 8, 3, 4, 2, 3, 6, 3, 1, 6, 5, 2, 6, 4, 5, 6, 2, 5, 1, 4, 2, 8, 6, 2, 2, 2, 8, 7, 8, 5, 7, 5, 7, 2, 5, 8, 4, 6, 5, 1, 4, 4, 3, 3, 6, 6 \right]$

Execution

➔ *Load* $Q = T[6] = P + \phi(P) + \psi(\phi(P))$

64 times

➔ $Q = 2Q - T[6] = P + \phi(P) + \psi(\phi(P))$

➔ $Q = 2Q + T[3] = 3P + 2\phi(P) + \psi(P) + 2\psi(\phi(P))$

➔ $Q = 2Q - T[5] = 5P + 4\phi(P) + 2\psi(P) + 3\psi(\phi(P))$

$\vdots$

T[1]	P
T[2]	$P + \phi(P)$
T[3]	$P + \psi(P)$
T[4]	$P + \phi(P) + \psi(P)$
T[5]	$P + \psi(\phi(P))$
T[6]	$P + \phi(P) + \psi(\phi(P))$
T[7]	$P + \psi(P) + \psi(\phi(P))$
T[8]	$P + \phi(P) + \psi(P) + \psi(\phi(P))$

- **Regular execution** (exactly 64 DBLS and 64 ADDs) facilitates protection against timing/SSCA attacks.
- Reduced number of precomputations (only 8 points).

Regular Multi-Scalar Multiplication

$$\begin{array}{ll} \text{column signs } s_i & [+ - + - + + - + - + - - - - + - + - + + - - - + - - + + + - - + + - - + + + + + + - - + + + + + - - - - + - + - - - - + - + - - -] \\ \text{digits } d_i & [\cancel{6}, \cancel{6}, 3, 5, 7, 6, 7, 3, 2, 2, 3, 2, 2, 1, 8, 1, 5, 1, 6, 8, 8, 3, 4, 2, 3, 6, 3, 1, 6, 5, 2, 6, 4, 5, 6, 2, 5, 1, 4, 2, 8, 6, 2, 2, 2, 8, 7, 8, 5, 7, 5, 7, 2, 5, 8, 4, 6, 5, 1, 4, 4, 3, 3, 6, 6] \end{array}$$

Execution

➔ *Load* $Q = T[6] = P + \phi(P) + \psi(\phi(P))$

64 times

➔ $Q = 2Q - T[6] = P + \phi(P) + \psi(\phi(P))$

➔ $Q = 2Q + T[3] = 3P + 2\phi(P) + \psi(P) + 2\psi(\phi(P))$

➔ $Q = 2Q - T[5] = 5P + 4\phi(P) + 2\psi(P) + 3\psi(\phi(P))$

$\vdots$

T[1]	P
T[2]	$P + \phi(P)$
T[3]	$P + \psi(P)$
T[4]	$P + \phi(P) + \psi(P)$
T[5]	$P + \psi(\phi(P))$
T[6]	$P + \phi(P) + \psi(\phi(P))$
T[7]	$P + \psi(P) + \psi(\phi(P))$
T[8]	$P + \phi(P) + \psi(P) + \psi(\phi(P))$

- **Regular execution** (exactly 64 DBLS and 64 ADDs) facilitates protection against timing/SSCA attacks.
- Reduced number of precomputations (only 8 points).

Regular Multi-Scalar Multiplication

$$\begin{array}{ll} \text{column signs } s_i & [+ - + - + + - + - + - - - - + - + - + + - - - + - - + + + - - + + - - + + + + + + - - + + + + + - - - - + - + - - - - + - + - - -] \\ \text{digits } d_i & [\cancel{6}, \cancel{6}, \cancel{3}, 5, 7, 6, 7, 3, 2, 2, 3, 2, 2, 1, 8, 1, 5, 1, 6, 8, 8, 3, 4, 2, 3, 6, 3, 1, 6, 5, 2, 6, 4, 5, 6, 2, 5, 1, 4, 2, 8, 6, 2, 2, 2, 8, 7, 8, 5, 7, 5, 7, 2, 5, 8, 4, 6, 5, 1, 4, 4, 3, 3, 6, 6] \end{array}$$

Execution

➔ *Load* $Q = T[6] = P + \phi(P) + \psi(\phi(P))$

64 times

➔ $Q = 2Q - T[6] = P + \phi(P) + \psi(\phi(P))$

➔ $Q = 2Q + T[3] = 3P + 2\phi(P) + \psi(P) + 2\psi(\phi(P))$

➔ $Q = 2Q - T[5] = 5P + 4\phi(P) + 2\psi(P) + 3\psi(\phi(P))$

$\vdots$

T[1]	P
T[2]	$P + \phi(P)$
T[3]	$P + \psi(P)$
T[4]	$P + \phi(P) + \psi(P)$
T[5]	$P + \psi(\phi(P))$
T[6]	$P + \phi(P) + \psi(\phi(P))$
T[7]	$P + \psi(P) + \psi(\phi(P))$
T[8]	$P + \phi(P) + \psi(P) + \psi(\phi(P))$

- **Regular execution** (exactly 64 DBLS and 64 ADDs) facilitates protection against timing/SSCA attacks.
- Reduced number of precomputations (only 8 points).

Regular Multi-Scalar Multiplication

column signs s_i $\left[\begin{array}{l} + - + - + + - + - + - - - - + - + - + + - - - + - - + + + - - + + - - + + + + + + - - + + + + + - - - - + - + - - - - + - + - - - \end{array}\right]$

digits d_i $\left[\cancel{6}, \cancel{6}, \cancel{3}, \cancel{5}, 7, 6, 7, 3, 2, 2, 3, 2, 2, 1, 8, 1, 5, 1, 6, 8, 8, 3, 4, 2, 3, 6, 3, 1, 6, 5, 2, 6, 4, 5, 6, 2, 5, 1, 4, 2, 8, 6, 2, 2, 2, 8, 7, 8, 5, 7, 5, 7, 2, 5, 8, 4, 6, 5, 1, 4, 4, 3, 3, 6, 6\right]$

Execution

➔ *Load* $Q = T[6] = P + \phi(P) + \psi(\phi(P))$

64 times

➔ $Q = 2Q - T[6] = P + \phi(P) + \psi(\phi(P))$

➔ $Q = 2Q + T[3] = 3P + 2\phi(P) + \psi(P) + 2\psi(\phi(P))$

➔ $Q = 2Q - T[5] = 5P + 4\phi(P) + 2\psi(P) + 3\psi(\phi(P))$

$\vdots$

T[1]	P
T[2]	$P + \phi(P)$
T[3]	$P + \psi(P)$
T[4]	$P + \phi(P) + \psi(P)$
T[5]	$P + \psi(\phi(P))$
T[6]	$P + \phi(P) + \psi(\phi(P))$
T[7]	$P + \psi(P) + \psi(\phi(P))$
T[8]	$P + \phi(P) + \psi(P) + \psi(\phi(P))$

- **Regular execution** (exactly 64 DBLS and 64 ADDs) facilitates protection against timing/SSCA attacks.
- Reduced number of precomputations (only 8 points).

Regular Multi-Scalar Multiplication

$$
\begin{array}{ll}
\text{column signs } s_i & [+ - + - + + - + - + - - - - + - + - + + - - - + - - + + + - - + + - + + - - + + + + + + - - + + + + + - - - - + - + - - - - + - + - - -] \\
\text{digits } d_i & [\cancel{6}, \cancel{6}, \cancel{3}, \cancel{5}, 7, 6, 7, 3, 2, 2, 3, 2, 2, 1, 8, 1, 5, 1, 6, 8, 8, 3, 4, 2, 3, 6, 3, 1, 6, 5, 2, 6, 4, 5, 6, 2, 5, 1, 4, 2, 8, 6, 2, 2, 2, 8, 7, 8, 5, 7, 5, 7, 2, 5, 8, 4, 6, 5, 1, 4, 4, 3, 3, 6, 6]
\end{array}
$$

Execution

➔ $Load\ \ Q = T[6] = P + \phi(P) + \psi(\phi(P))$

64 times

➔ $Q = 2Q - T[6] = P + \phi(P) + \psi(\phi(P))$

➔ $Q = 2Q + T[3] = 3P + 2\phi(P) + \psi(P) + 2\psi(\phi(P))$

➔ $Q = 2Q - T[5] = 5P + 4\phi(P) + 2\psi(P) + 3\psi(\phi(P))$

$\vdots$

T[1]	P
T[2]	$P + \phi(P)$
T[3]	$P + \psi(P)$
T[4]	$P + \phi(P) + \psi(P)$
T[5]	$P + \psi(\phi(P))$
T[6]	$P + \phi(P) + \psi(\phi(P))$
T[7]	$P + \psi(P) + \psi(\phi(P))$
T[8]	$P + \phi(P) + \psi(P) + \psi(\phi(P))$

- **Regular execution** (exactly 64 DBLS and 64 ADDs) facilitates protection against timing/SSCA attacks.
- Reduced number of precomputations (only 8 points).

SPA/DPA-protected scalar multiplication

SPA countermeasures

- Constant-time, constant-flow implementations
  - ✓Complete formulas
  - ✓Ladder-based or regular double-and-add based algorithms

SPA/DPA-protected scalar multiplication

SPA countermeasures

- Constant-time, constant-flow implementations
  - ✓Complete formulas
  - ✓Ladder-based or regular double-and-add based algorithms

Previous protections do not prevent

- **Differential Power Analysis (DPA)**: many traces with same key and varying plaintext
- Other variants: template attacks: very powerful attacker

DPA countermeasures for scalar multiplication

- 1999: J.S. Coron suggested **randomizing the computation** by using:
  1. **Scalar randomization**

$$m \cdot P \equiv (m + r \cdot \#E) \cdot P$$

DPA countermeasures for scalar multiplication

- 1999: J.S. Coron suggested **randomizing the computation** by using:
  1. **Scalar randomization**

     $$m \cdot P \equiv (m + r \cdot \#E) \cdot P$$

  2. **Base point blinding (inspired by Chaum's blind signatures)**

     Blind: for random R

     $$\widetilde{\boldsymbol{P}} \leftarrow P + R$$

     Scalar multiplication:

     $$Q \leftarrow m \cdot \widetilde{\boldsymbol{P}}$$

     Unblind:

     $$P = Q - m \cdot R$$

     Moreover, update R for the next scalar multiplication.

     $$R \leftarrow (-1)^b 2R$$

DPA countermeasures for scalar multiplication

- 1999: J.S. Coron suggested **randomizing the computation** by using:
  1. **Scalar randomization**
  $$m \cdot P \equiv (m + r \cdot \#E) \cdot P$$
  2. **Base point blinding (inspired by Chaum's blind signatures)**
  Blind: for random R
  $$\widetilde{\boldsymbol{P}} \leftarrow P + R$$
  Scalar multiplication:
  $$Q \leftarrow m \cdot \widetilde{\boldsymbol{P}}$$
  Unblind:
  $$P = Q - m \cdot R$$
  Moreover, update R for the next scalar multiplication.
  $$R \leftarrow (-1)^b 2R$$
  3. **Projective coordinates randomization**
  $$P = (X:Y:Z) \equiv (\lambda X:\lambda Y:\lambda Z), \text{ for random } \lambda \neq 0$$

Scalar randomization

- 1999: J.S. Coron suggested scalar randomization

$$m \cdot P \equiv (m + r \cdot \#E) \cdot P$$

  where r is small (e.g., 20 bits).

Scalar randomization

- 1999: J.S. Coron suggested scalar randomization

$$m \cdot P \equiv (m + r \cdot \#E) \cdot P$$

  where r is small (e.g., 20 bits).

- **Problem**: prime-order curves over pseudo-Mersenne primes

$$p = 2^{k_1} \pm 2^{k_2} \cdots + c\ ,$$

  present undesired repeated 1/0 patterns in $\#E$.

- **Unsafe** example: curve P-256:

$$\#E = 0x\mathbf{FFFFFFFF00000000FFFFFFFFFFFFFFFF}BCE6FAADA7179E84F3B9CAC2FC632551$$

$$r \cdot \#E = 0xC457\mathbf{FFFF}3BA8\mathbf{0000}C457\mathbf{FFFFFFFFFFFF}CC89C7531F9F857C484E071AFC42AAA6D7D80$$

$$m = 0xD31280\mathbf{4752}9A2D4\mathbf{BD63D96EB1D9}C4AC781588CFEFCFC153398F5ED03506AA58B$$

$$m + r \cdot \#E = 0xC4580D3063AC\mathbf{752}A672C\mathbf{BD63D96EB1D9}91363F68A86F754C09A140AA5B12DFAD8230B$$

DPA countermeasures for scalar multiplication

- **Safe** example: curve Four$\mathbb{Q}$: non-prime order, $\#E = 392 * N$

$$\#E = 0x3\textcolor{red}{FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF}E9968776D07B232910910B6A16D87C1B8$$

DPA countermeasures for scalar multiplication

- **Safe** example: curve Four$\mathbb{Q}$: non-prime order, $\#E = 392 * N$

$$\#E = 0x3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE9968776D07B232910910B6A16D87C1B8$$

But we usually work with the prime-order **subgroup** where $\#P = N$, therefore

$$m \cdot P \equiv (m + r \cdot \#N) \cdot P$$

and notice that

$$N = \mathbf{0x29CBC14E5E0A72F05397829CBC14E5DFBD004DFE0F79992FB2540EC7768CE7}$$

does not present the undesired patterns and Coron's technique **could be used**.

DPA countermeasures for scalar multiplication

- **Safe** example: curve Four$\mathbb{Q}$: non-prime order, $\#E = 392 * N$

  $\#E = \mathit{0x3}\textcolor{red}{\mathit{FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF}}\mathit{E9968776D07B232910910B6A16D87C1B8}$

  But we usually work with the prime-order **subgroup** where $\#P = N$, therefore

  $$m \cdot P \equiv (m + r \cdot \#N) \cdot P$$

  and notice that

  $N = \boldsymbol{\mathit{0x29CBC14E5E0A72F05397829CBC14E5DFBD004DFE0F79992FB2540EC7768CE7}}$

  does not present the undesired patterns and Coron's technique **could be used**.
- Can we do better in Four$\mathbb{Q}$? A.: yes.

Scalar randomization

- **Remark**: Coron’s method is inefficient for curves with endomorphisms.

- In **FourQ**, we extended to Ciet et al.’s GLV scalar randomization
  - Extend every mini-scalar by 16 bits (64 bits in total)
  - No problem with pattern repetitions
  - Overhead is only 25% (compared against at least 50% overhead in curve25519)

Side-channel protected Four$\mathbb{Q}$

Algorithm 2. SCA-protected Four$\mathbb{Q}$'s scalar multiplication on $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

Input: Point $P = (x_P, y_P)$, blinding point $R = (x_R, y_R) \in \mathcal{E}(\mathbb{F}_{p^2})[N]$, integer scalar m and random value $s \in [0, 2^{256})$, a random bit b, and random values $[r_{81}, r_{80}, \dots, r_0] \in \mathbb{F}_p^{82}$.

Output: $[m]P$ and updated point R.

Randomize input points and update blinding point R:

Set $R = (r_{81} \cdot x_R, r_{81} \cdot y_R, r_{81})$.

Compute $R = [(-1)^b 3]R$.

Set $P = (r_{80} \cdot x_P, r_{80} \cdot y_P, r_{80})$.

Compute endomorphisms and precompute lookup table:

Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$.

Compute $T[u] = -R + [u_0]P + [u_1]\phi(P) + [u_2]\psi(P) + [u_3]\psi(\phi(P))$ for $u = (u_3, u_2, u_1, u_0)_2$ in $0 \le u \le 15$. Write $T[u]$ in coordinates (X, Y, Z).

Scalar decomposition, randomization and recoding:

Decompose m into the multiscalar (a_1, a_2, a_3, a_4) as in [16, Prop. 5].

Randomize (a_1, a_2, a_3, a_4) as in Proposition 1 and recode to digit-columns $(d_{79}, \dots, d_0)$ s.t. $d_i = a_1[i] + 2a_2[i] + 4a_3[i] + 8a_4[i]$ for $i = 0, \dots, 79$.

Main loop:

```
Q = R
for i = 79 to 0 do
    S = (r_i · X_{T[d_i]}, r_i · Y_{T[d_i]}, r_i · Z_{T[d_i]}).
    Q = [2]Q + S
return (Q − R) and R in affine coordinates.
```

Side-channel protected Four$\mathbb{Q}$

Algorithm 2. SCA-protected Four$\mathbb{Q}$'s scalar multiplication on $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

Input: Point $P = (x_P, y_P)$, blinding point $R = (x_R, y_R) \in \mathcal{E}(\mathbb{F}_{p^2})[N]$, integer scalar m and random value $s \in [0, 2^{256})$, a random bit b, and random values $[r_{81}, r_{80}, \ldots, r_0] \in \mathbb{F}_p^{82}$.

Output: $[m]P$ and updated point R.

Randomize input points and update blinding point R:

1: Set $R = (r_{81} \cdot x_R, r_{81} \cdot y_R, r_{81})$.

2: Compute $\boxed{R = [(-1)^b 3]R.}$

3: Set $P = (r_{80} \cdot x_P, r_{80} \cdot y_P, r_{80})$.

Compute endomorphisms and precompute lookup table:

4: Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$.

5: Compute $T[u] = -R + [u_0]P + [u_1]\phi(P) + [u_2]\psi(P) + [u_3]\psi(\phi(P))$ for $u = (u_3, u_2, u_1, u_0)_2$ in $0 \le u \le 15$. Write $T[u]$ in coordinates (X, Y, Z).

Scalar decomposition, randomization and recoding:

6: Decompose m into the multiscalar (a_1, a_2, a_3, a_4) as in [16, Prop. 5].

7: Randomize (a_1, a_2, a_3, a_4) as in Proposition 1 and recode to digit-columns $(d_{79}, \ldots, d_0)$ s.t. $d_i = a_1[i] + 2a_2[i] + 4a_3[i] + 8a_4[i]$ for $i = 0, ..., 79$.

Main loop:

8: $Q = R$

9: **for** $i = 79$ **to** 0 **do**

10: $\quad S = (r_i \cdot X_{T[d_i]}, r_i \cdot Y_{T[d_i]}, r_i \cdot Z_{T[d_i]})$.

11: $\quad Q = [2]Q + S$

12: **return** $(Q - R)$ and R in affine coordinates.

Always update blinding point R

Side-channel protected Four$\mathbb{Q}$

Algorithm 2. SCA-protected Four$\mathbb{Q}$'s scalar multiplication on $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

Input: Point $P = (x_P, y_P)$, blinding point $R = (x_R, y_R) \in \mathcal{E}(\mathbb{F}_{p^2})[N]$, integer scalar m and random value $s \in [0, 2^{256})$, a random bit b, and random values $[r_{81}, r_{80}, \ldots, r_0] \in \mathbb{F}_p^{82}$.

Output: $[m]P$ and updated point R.

Randomize input points and update blinding point R:

1: Set $R = (r_{81} \cdot x_R, r_{81} \cdot y_R, r_{81})$.

2: Compute $R = [(-1)^b 3]R$.

3: Set $P = (r_{80} \cdot x_P, r_{80} \cdot y_P, r_{80})$.

Compute endomorphisms and precompute lookup table:

4: Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$.

5: Compute $T[u] = \boxed{-R} + [u_0]P + [u_1]\phi(P) + [u_2]\psi(P) + [u_3]\psi(\phi(P))$ for $u = (u_3, u_2, u_1, u_0)_2$ in $0 \le u \le 15$. Write $T[u]$ in coordinates (X, Y, Z).

Scalar decomposition, randomization and recoding:

6: Decompose m into the multiscalar (a_1, a_2, a_3, a_4) as in [16, Prop. 5].

7: Randomize (a_1, a_2, a_3, a_4) as in Proposition 1 and recode to digit-columns $(d_{79}, \ldots, d_0)$ s.t. $d_i = a_1[i] + 2a_2[i] + 4a_3[i] + 8a_4[i]$ for $i = 0, \ldots, 79$.

Main loop:

8: $Q = R$

9: **for** $i = 79$ **to** 0 **do**

10: $S = (r_i \cdot X_{T[d_i]}, r_i \cdot Y_{T[d_i]}, r_i \cdot Z_{T[d_i]})$.

11: $Q = [2]Q + S$

12: **return** $(Q - R)$ and R in affine coordinates.

Blinding point R plays a role in T
'Sign-alignment' cannot be used here, thus
New table has now 16 points

Side-channel protected FourQ

Algorithm 2. SCA-protected FourQ's scalar multiplication on $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

Input: Point $P = (x_P, y_P)$, blinding point $R = (x_R, y_R) \in \mathcal{E}(\mathbb{F}_{p^2})[N]$, integer scalar m and random value $s \in [0, 2^{256})$, a random bit b, and random values $\boxed{[r_{81}, r_{80}, \ldots, r_0]} \in \mathbb{F}_p^{82}$.

projective coordinate randomization

Output: $[m]P$ and updated point R.

Randomize input points and update blinding point R:

Set $R = (r_{81} \cdot x_R, r_{81} \cdot y_R, r_{81})$.

Compute $R = [(-1)^b 3]R$.

Set $P = (r_{80} \cdot x_P, r_{80} \cdot y_P, r_{80})$.

Compute endomorphisms and precompute lookup table:

Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$.

Compute $T[u] = -R + [u_0]P + [u_1]\phi(P) + [u_2]\psi(P) + [u_3]\psi(\phi(P))$ for $u = (u_3, u_2, u_1, u_0)_2$ in $0 \le u \le 15$. Write $T[u]$ in coordinates (X, Y, Z).

Scalar decomposition, randomization and recoding:

Decompose m into the multiscalar (a_1, a_2, a_3, a_4) as in [16, Prop. 5].

Randomize (a_1, a_2, a_3, a_4) as in Proposition 1 and recode to digit-columns $(d_{79}, \ldots, d_0)$ s.t. $d_i = a_1[i] + 2a_2[i] + 4a_3[i] + 8a_4[i]$ for $i = 0, \ldots, 79$.

Main loop:

$Q = R$

for $i = 79$ **to** 0 **do**

  $S = (r_i \cdot X_{T[d_i]}, r_i \cdot Y_{T[d_i]}, r_i \cdot Z_{T[d_i]})$.

  $Q = [2]Q + S$

return $(Q - R)$ and R in affine coordinates.

Side-channel protected FourQ

Algorithm 2. SCA-protected FourQ's scalar multiplication on $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

Input: Point $P = (x_P, y_P)$, blinding point $R = (x_R, y_R) \in \mathcal{E}(\mathbb{F}_{p^2})[N]$, integer scalar m and random value $s \in [0, 2^{256})$, a random bit b, and random values $\boxed{[r_{81}, r_{80}, \ldots, r_0]} \in \mathbb{F}_p^{82}$.

Output: $[m]P$ and updated point R.

Randomize input points and update blinding point R:

Set $\boxed{R = (r_{81} \cdot x_R, r_{81} \cdot y_R, r_{81})}$

Compute $R = [(-1)^b 3]R$.

Set $\boxed{P = (r_{80} \cdot x_P, r_{80} \cdot y_P, r_{80})}$

Compute endomorphisms and precompute lookup table:

Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$.

Compute $T[u] = -R + [u_0]P + [u_1]\phi(P) + [u_2]\psi(P) + [u_3]\psi(\phi(P))$ for $u = (u_3, u_2, u_1, u_0)_2$ in $0 \le u \le 15$. Write $T[u]$ in coordinates (X, Y, Z).

Scalar decomposition, randomization and recoding:

Decompose m into the multiscalar (a_1, a_2, a_3, a_4) as in [16, Prop. 5].

Randomize (a_1, a_2, a_3, a_4) as in Proposition 1 and recode to digit-columns $(d_{79}, \ldots, d_0)$ s.t. $d_i = a_1[i] + 2a_2[i] + 4a_3[i] + 8a_4[i]$ for $i = 0, \ldots, 79$.

Main loop:

$Q = R$

for $i = 79$ **to** 0 **do**

  $\boxed{S = (r_i \cdot X_{T[d_i]}, r_i \cdot Y_{T[d_i]}, r_i \cdot Z_{T[d_i]})}$

  $Q = [2]Q + S$

return $(Q - R)$ and R in affine coordinates.

projective coordinate randomization

Side-channel protected Four$\mathbb{Q}$

Algorithm 2. SCA-protected Four$\mathbb{Q}$'s scalar multiplication on $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

Input: Point $P = (x_P, y_P)$, blinding point $R = (x_R, y_R) \in \mathcal{E}(\mathbb{F}_{p^2})[N]$, integer scalar m and random value $s \in [0, 2^{256})$, a random bit b, and random values $[r_{81}, r_{80}, \ldots, r_0] \in \mathbb{F}_p^{82}$.

Output: $[m]P$ and updated point R.

Randomize input points and update blinding point R:

Set $R = (r_{81} \cdot x_R, r_{81} \cdot y_R, r_{81})$.

Compute $R = [(-1)^b 3]R$.

Set $P = (r_{80} \cdot x_P, r_{80} \cdot y_P, r_{80})$.

Compute endomorphisms and precompute lookup table:

Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$.

Compute $T[u] = -R + [u_0]P + [u_1]\phi(P) + [u_2]\psi(P) + [u_3]\psi(\phi(P))$ for $u = (u_3, u_2, u_1, u_0)_2$ in $0 \le u \le 15$. Write $T[u]$ in coordinates (X, Y, Z).

Scalar decomposition, randomization and recoding:

Decompose m into the multiscalar (a_1, a_2, a_3, a_4) as in [16, Prop. 5].

$\boxed{\text{Randomize}}$ (a_1, a_2, a_3, a_4) as in Proposition 1 and recode to digit-columns $(d_{79}, \ldots, d_0)$ s.t. $d_i = a_1[i] + 2a_2[i] + 4a_3[i] + 8a_4[i]$ for $i = 0, \ldots, 79$.

Main loop:

$Q = R$

for $\boxed{i = 79 \textbf{ to } 0 \textbf{ do}}$

$\quad S = (r_i \cdot X_{T[d_i]}, r_i \cdot Y_{T[d_i]}, r_i \cdot Z_{T[d_i]})$.

$\quad Q = [2]Q + S$

return $(Q - R)$ and R in affine coordinates.

Multi-scalar randomization adds 16 bits

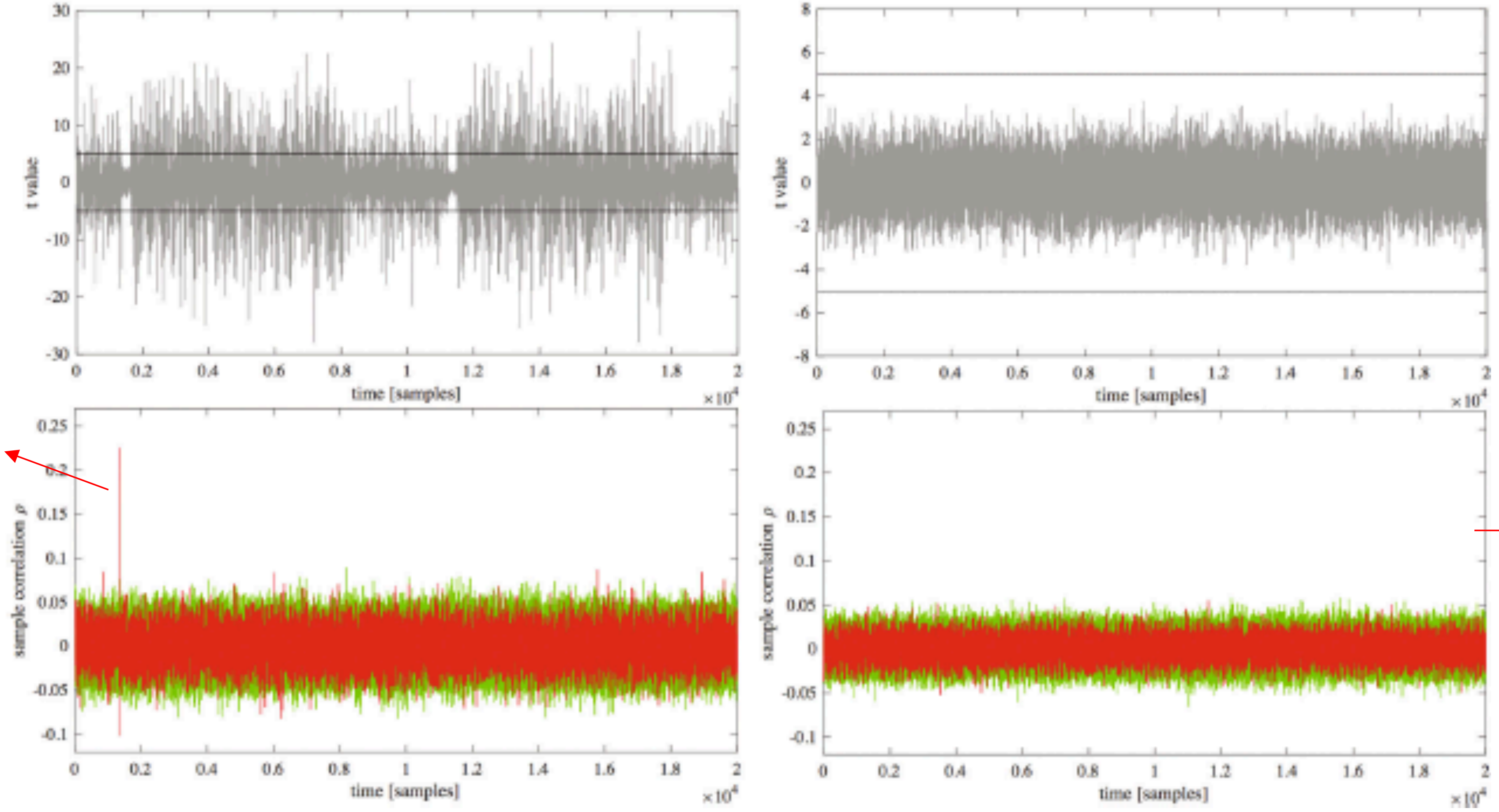
Slightly larger loop length (~~64~~ -> 80)

Side-channel evaluation

- Carried out a practical side-channel evaluation on an ARM Cortex-M4 with no dedicated security features.
- EM traces. Low noise: DPA with a dozen measurements works.
- Performed leakage detection and key-recovery attacks for vertical DPA attacks
- Tested the effectiveness of each countermeasure first in isolation and then combined
- **No leakage detected with up to 10 million measurements with all countermeasures activated**

Side-channel evaluation: **point blinding correlation**

Target correlation detected



No first-order leakage detected

All countermeasures disabled

Only point blinding enabled

FourQ software for embedded systems

- Open-source (MIT license).
- C language + Assembly (optional)
- ARM Cortex M4 (32-bit), MSP430(X) (16-bit), AVR ATxmega (8-bit)
- Highly customizable:
  - w/ or w/o endomorphisms, tables sizes, w/ or w/o assembly
- Crypto primitives
  - KeyAgreement (w/ and w/o compression)
  - [Update] SchnorrQ signature recently included (extended version)
- Speed-records set for ECDH and signatures.

Speed-record results (speed prioritized)

Source	Scalar multiplication		ECDH	
	Fixed-base	Random	Static	Ephemeral
8-bit AVR ATmega				
Curve25519	13,900,400	13,900,400	13,900,400	27,800,800
μKummer	9,513,500	9,513,500	9,739,100	19,027,100
FourQ (this work)	**2,980,700**	**6,505,300**	**6,886,400** **7,221,300**	**9,870,500** **10,206,500**
16-bit MSP430X (16-bit multiplier) @8 MHz				
Curve25519	7,933,300	7,933,300	7,933,300	15,866,600
FourQ (this work)	**1,851,300**	**4,280,400**	**4,527,900** **4,826,100**	**6,379,200** **6,677,400**
32-bit ARM Cortex-M4				
Curve25519	1,423,700	1,423,700	1,423,700	2,847,400
FourQ (this work)	**232,900**	**469,500**	**496,400** **542,900**	**729,900** **776,600**

Speed-record results (speed prioritized)

Source	Scalar multiplication		ECDH	
	Fixed-base	Random	Static	Ephemeral
8-bit AVR ATmega				
Curve25519	13,900,400	13,900,400	13,900,400	27,800,800
μKummer	9,513,500	9,513,500	9,739,100	19,027,100
FourQ (this work)	**2,980,700**	**6,505,300**	**6,886,400**	**9,870,500**
			7,221,300	**10,206,500**
16-bit MSP430X (16-bit multiplier) @8 MHz				
Curve25519	7,933,300	7,933,300	7,933,300	15,866,600
FourQ (this work)	**1,851,300**	**4,280,400**	**4,527,900**	**6,379,200**
			4,826,100	**6,677,400**
32-bit ARM Cortex-M4				
Curve25519	1,423,700	1,423,700	1,423,700	2,847,400
FourQ (this work)	**232,900**	**469,500**	**496,400**	**729,900**
			542,900	**776,600**

Renes'16

1.9x

Speed-record results (speed prioritized)

Source	Scalar multiplication		ECDH	
	Fixed-base	Random	Static	Ephemeral
8-bit AVR ATmega				
Curve25519	13,900,400	13,900,400	13,900,400	27,800,800
μKummer	9,513,500	9,513,500	9,739,100	19,027,100
FourQ (this work)	**2,980,700**	**6,505,300**	**6,886,400** **7,221,300**	**9,870,500** **10,206,500**
16-bit MSP430X (16-bit multiplier) @8 MHz				
Curve25519	7,933,300	7,933,300	7,933,300	15,866,600
FourQ (this work)	**1,851,300**	**4,280,400**	**4,527,900** **4,826,100**	**6,379,200** **6,677,400**
32-bit ARM Cortex-M4				
Curve25519	1,423,700	1,423,700	1,423,700	2,847,400
FourQ (this work)	**232,900**	**469,500**	**496,400** **542,900**	**729,900** **776,600**

Düll'15

2.8x

2.5x

3.9x

Remarks and future work

- *Fast and secure state-of-the-art implementation of Four$\mathbb{Q}$ on embedded devices*
- *Proof of concept:* ***open-source library*** *+* ***side-channel evaluation***
  - ***https://github.com/Microsoft/FourQlib***
  - ***https://github.com/geovandro/microFourQ-AVR***
  - ***https://github.com/geovandro/microFourQ-MSP***
- *Focused on speed*
  - *Would be interesting to analyze* ***memory tradeoffs***
- *Would also be interesting to extend to* ***other languages*** *(Javascript, Rust) and* ***different platforms****.*

Four$\mathbb{Q}$ on Embedded Devices with Strong Countermeasures Against Side-Channel Attacks

Geovandro C. C. F. Pereira

geovandro.pereira@uwaterloo.ca

UNIVERSITY OF WATERLOO | IQC Institute for Quantum Computing