On the Implementation Challenges of Multi-Scalar-Multiplication for SNARKs

Raphaël Comps[1], Viktor Fischer[2], Carlos Lara[2], Vladimír Marcin[1], Tibor Tribus[1]

19th International Workshop on Cryptographic Architectures Embedded in Logic Devices

Castro Urdiales, Spain

June 13, 2023

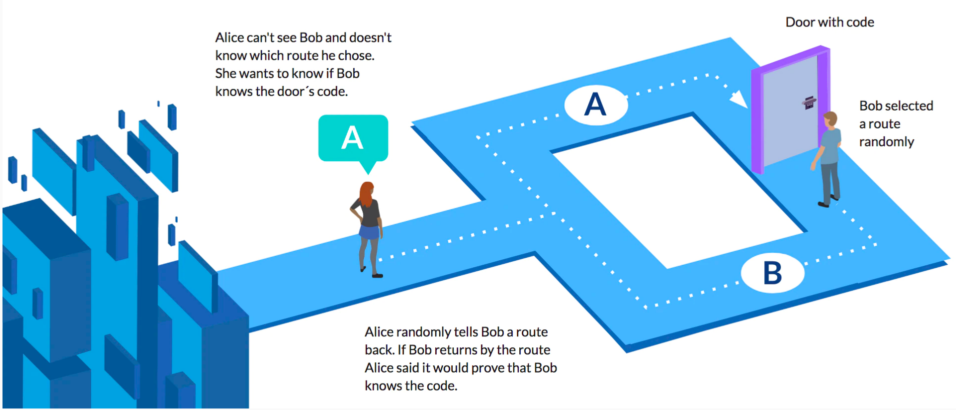
1. MAYA-ZK, PRAHA, Česká republika
2. Université Jean Monnet Saint-Etienne, CNRS, Laboratoire Hubert Curien
UMR 5516, F-42023, SAINT-ETIENNE, France

Motivation

- A method by which one party (the prover) can prove to another party (the verifier) that a given statement is true.
- The prover does not convey any additional information apart from the fact that the statement is indeed true.
- While it is trivial to prove that one possesses knowledge of certain information by simply revealing it, the challenge is to prove such possession without revealing the information itself or any additional information.

- Verify an individual's identity, without revealing any sensitive personal information.
- Create voting mechanisms that enable individuals to cast votes without compromising their identity or revealing who they voted for.
- Enable blockchain nodes to validate transactions without needing to access transaction data.

ZK in a nutshell



[1] [2]

[1] Quisquater et al. (2001). How to explain zero-knowledge protocols to your children. In *CRYPTO'89* (pp. 628-631). Springer New York.

[2] Illustration retrieved from `www.bbva.com`

ZK protocols

A broad ZKP classification

Based on their underlying operations :

- proof of knowledge
- witness indistinguishable proof
- multi-party computation
- ring signatures
- polynomial commitments
  - Succinct Non-Interactive ARgument of Knowledge (SNARK)
  - Scalable Transparent ARgument of Knowledge (STARK)

Commitment schemes

1. Commit

Wait for all votes to be committed....

2. Reveal

Count votes and declare winner!

Polynomial commitments

Commitment schemes:

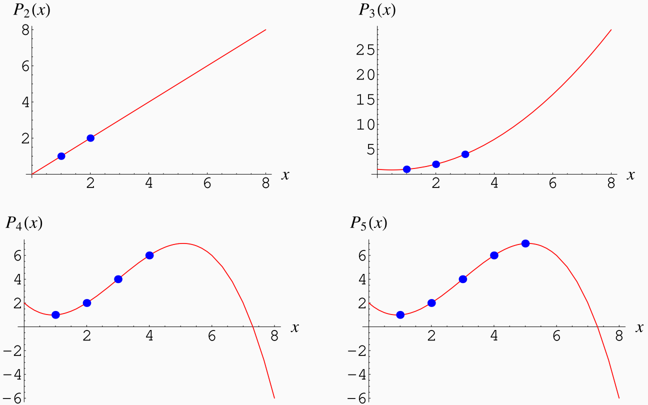
- **binding** : once publishing a commitment c, the committer should not be able to find some other message $m' \neq m$ which also corresponds to c
- **hiding** : publishing c should not reveal any information about m

Polynomial commitments:

- **incremental** : the committer should be able to “open” certain evaluations of the committed polynomial without revealing the entire thing

Background on polynomials

In general, it's possible to take n arbitrary points and find a unique polynomial of degree $n-1$ which passes through all of them. This process is called "polynomial interpolation."



https://mathworld.wolfram.com/

A polynomial $P(x) = \sum_{i=0}^{n1} p_i x_i$ of degree $(n-1)$ can be represented in two ways:

- Coefficient form :
  $P(x)$ can be represented as a tuple of its n coefficients: $[p_0, p_1, \ldots, p_{n-1}]$
- Evaluation form :
  $P(x)$ can be represented as a tuple of n disctinct evaluations: $[P(x_0), P(x_1), \ldots, P(x_{n-1})]$

Converting from coefficient form to evaluation form is analogous to solving a Fourier transform, and converting in the reverse direction is analogous to solving an inverse Fourier transform.

Naively : from coefficient form, we can simply evaluate the polynomial at each x_i in evaluation domain. From evaluation form, we can use Lagrange interpolation to obtain the unique degree $(n1)$ polynomial passing through each of the n points.

Permutations over **L**agrange-bases for **O**ecumenical **N**oninteractive arguments of **K**nowledge

- **Phase 1 :** Write out the "witness"
  - The *witness* refers to some data that shows why a statement is true.

A	B	C	S	P	Z
f_0	f_1	f_2	1	f_0	$z(\omega_0)$
f_1	f_2	f_3	1	f_1	$z(\omega_1)$
f_2	f_3	f_4	1	k	$z(\omega_2)$
...	...	...	...	...	...
f_{n-3}	f_{n-2}	f_{n-1}	1		$z(\omega_{n-3})$
f_{n-2}	f_{n-1}	f_n	1		$z(\omega_{n-2})$
			0		$z(\omega_{n-1})$

The trace table is a 2-dimensional matrix where the *witness* is written down. It also includes other values that are useful in demonstrating that the witness is correct. Each cell is an element of a large finite field.

Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge

- **Phase 2 :** Commit to the trace table
  - Create some succinct representation of the *witness*, compressing it.
  - Using polynomial commitments allows to prove certain properties about the original *witness*, referencing just the succinct commitment.

A	B	C	S	P	Z
$f_0 = a(\alpha_0)$	$f_1 = b(\beta_0)$	$f_2 = c(\gamma_0)$	$1 = s(\sigma_0)$	$f_0 = p(\rho_0)$	$z(\omega_0)$
$f_1 = a(\alpha_1)$	$f_2 = b(\beta_1)$	$f_3 = c(\gamma_1)$	$1 = s(\sigma_1)$	$f_1 = p(\rho_1)$	$z(\omega_1)$
$f_2 = a(\alpha_2)$	$f_3 = b(\beta_2)$	$f_4 = b(\gamma_2)$	$1 = s(\sigma_2)$	$k_0 = p(\rho_2)$	$z(\omega_2)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$f_{n-3} = a(\alpha_{n-3})$	$f_{n-2} = b(\beta_{n-3})$	$f_{n-1} = c(\gamma_{n-3})$	$1 = s(\sigma_{n-3})$	$k_{n-5} = p(\rho_{n-3})$	$z(\omega_{n-3})$
$f_{n-2} = a(\alpha_{n-2})$	$f_{n-1} = b(\beta_{n-2})$	$f_n = c(\gamma_{n-2})$	$1 = s(\sigma_{n-2})$	$k_{n-4} = p(\rho_{n-2})$	$z(\omega_{n-2})$
$a(\alpha_{n-1})$	$b(\beta_{n-1})$	$c(\gamma_{n-1})$	$0 = s(\sigma_{n-1})$	$p(\rho_{n-1})$	$z(\omega_{n-1})$

Consider column A from the trace table. This column is simply a length-n vector of finite field elements. We can think of this vector as the evaluation form of a unique polynomial $a(\alpha)$ with degree $(n1)$: the i_{th} element of A corresponds to the evaluation $a(\alpha_i)$.

Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge

- **Phase 2 :** Commit to the trace table
  - Use polynomial commitments to "compress" each column into a short representation
  - This also allows to generate proofs of evaluation : the prover can convince a verifier that the polynomial passes through a particular point, without revealing the entire polynomial.

Trusted setup :

$$\langle G \rangle = EC(\mathbb{F}_q)$$

$$l \in \mathbb{Z}$$

$$\tau \in \mathbb{F}_p$$

$$(G, \tau \cdot G, \tau^2 \cdot G, \ldots, \tau^l \cdot G)$$

Computing the commitment :

$$A(\alpha) \rightarrow A(\tau) \cdot G$$

$$A(\tau) \cdot G = \sum_{i=0}^{n-1} a_i \times \tau^i \cdot G$$

Lagrange-bases polynomials :

$$\ell_i(\chi) := \prod_{j \neq i} \frac{\chi - \chi_j}{\chi_i - \chi_j} \Bigg|_{i=0}^{n}$$

$$A(\tau) = \sum_{i=0}^{n-1} a(\alpha_i) \times \ell_i(\tau)$$

$$A(\alpha) = \sum_{i=0}^{n-1} a(\alpha_i) \times \ell_i(\tau) \cdot G$$

Permutations over **L**agrange-bases for **O**ecumenical **N**oninteractive arguments of **K**nowledge

- **Phase 3 :** "Prove" that the *witness* is correct
  - The witness generated in phase 1 must obey certain properties to be valid.
  - A short proof that the original witness satisfies these properties can be generated.

Let $\quad s(\sigma_i) \times f_k(a(\alpha_i), b(\beta_i), c(\gamma_i)) = 0\Big|_{i=0}^{n-1} \quad \Rightarrow \quad S(\sigma) \times f_k(A(\alpha), B(\beta), C(\gamma)) = 0 \quad$ (call it $\phi_k(\chi)$)

Suppose $k \in [0 \ldots m)$ and $\psi \in \mathbb{F}_q \quad \Rightarrow \quad \phi(\chi) := \psi^0 \times \phi_0(\chi) + \psi^1 \times \phi_1(\chi) + \ldots + \psi^{m-1} \times \phi_{m-1}(\chi)$

$\phi(\chi_i) = 0\Big|_{i=0}^{n-1} \quad \Leftrightarrow \quad \exists \quad \xi(\chi) \quad$ s.t. $\quad \phi(\chi) = \xi(\chi) \times (\chi^n - 1)$

$$\xi(\chi) := \frac{\phi(\chi)}{\chi^n - 1} = \frac{\psi^0 \times \phi_0(\chi) + \psi^1 \times \phi_1(\chi) + \ldots + \psi^{m-1} \times \phi_{m-1}(\chi)}{\chi^n - 1}$$

The prover need to solve a few number theoretic transforms and more multi-scalar multiplications.

The verifier selects χ and verifies if $\xi(\chi)$ holds $\quad \square$

Software results (2020):

	SNARK	STARK
Algorithmic complexity : prover	$\mathcal{O}(N \log N)$	$\mathcal{O}(N \log^k N)$
Algorithmic complexity : verifier	$\mathcal{O}(1)$	$\mathcal{O}(\log^k N)$
Communications costs (proof size)	$\mathcal{O}(1)$	$\mathcal{O}(\log^k N)$
Prover time	2.3s	1.6s
Verification time	10ms	16ms
Trusted setup required	Yes	No
Post-quantum secure	No	Maybe
Cryptographic fundamental	Bilinear pairings	Hashes
Complex computations	80% MSM[3], 15% NTT[4]	95% NTT

[3] Luo, G., Fu, S., & Gong, G. (2023). Speeding Up Multi-Scalar Multiplication over Fixed Points Towards Efficient zkSNARKs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 358-380.

[4] Chung, C. M. M., Hwang, V., Kannwischer, M. J., Seiler, G., Shih, C. J., & Yang, B. Y. (2021). NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 159-188.

Multi-Scalar Multiplication

- BLS12-377 curve
  - Let $a = 0$ and $b = 1$
  - $E : y^2 = x^3 + 1$ is defined over $\mathbb{F}_p$ with $|p| = 377$ and $\#EC(\mathbb{F}_p) = 256$
  - Birationally equivalent to the Montgomery curve

$$M : sy^2 = x^3 + 3\alpha sx^2 + x \quad \text{where} \quad \alpha^3 + a\alpha + b = 0 \quad \text{and} \quad s = \frac{1}{\sqrt{3\alpha^2 + a}}$$

  - Birationally equivalent to the Twisted Edards curve

$$T : ax^2 + y^2 = 1 + dx^2y^2 \quad \text{where} \quad a = \frac{3\alpha s + 2}{s} \quad \text{and} \quad d = \frac{3\alpha s - 2}{s}$$

MSM is the time-critical operation of SNARKs.

Solve

$$k_0 \cdot P_0 + k_1 \cdot P_1 + k_2 \cdot P_2 + k_3 \cdot P_3 + \ldots + k_{n-1} \cdot P_{n-1} = \sum_{i=0}^{n-1} k_i \cdot P_i$$

Pippenger's algorithm[5] allows to reduce the complexity of this operation and it only requires elliptic curve point addition.

- Partition each k_i into m parts such that each segment consists of c bits and $m = \lceil |p|/c \rceil$

$$\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} 2^{jc} \times k_i^j \cdot P_i = \sum_{j=0}^{m-1} 2^{jc} \sum_{i=0}^{n-1} k_i^j \cdot P_i = \sum_{j=0}^{m-1} 2^{jc} \times [k]P$$

- If we compute $[k]P$ for each k the last step can be solved through Horner's rule :

$$\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} 2^{jc} \times k_i^j \cdot P_i = 2^c(\ldots(2^c(2^c \times [k-1]\cdot P + [k-2]\cdot P) + [k-3]\cdot P)\ldots) + [0]\cdot P$$

- Thus the MSM is reduced to computing a series of elliptic curve point accumulations.

[5] Pippenger, N. (1976). On the evaluation of powers and related problems. In *17th Annual Symposium on Foundations of Computer Science* (pp. 258-263). IEEE Computer Society.

Point addition

- For a generic elliptic curve $E : y^2 = x^3 + 1$
  - Let $(x_1, y_1) \in E(\mathbb{F}_p)$ and $(x_2, y_2) \in E(\mathbb{F}_p)$
  - The addition of these two points is given by :

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \qquad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$$

  - Using projective representation $(x_i, y_i) \to (X_i, Y_i, Z_i)$:

$$T_2 = Z_1 * Y_2 \quad T_1 = Z_2 * Y_1 \quad T = T_2 - T_1 \quad U_2 = Z_1 * X_2 \quad U_1 = Z_2 * X_1 \quad U = U_2 - U_1$$

$$U_0 = U * U \quad V = Z_1 * Z_2 \quad W = T * T * V - U_0 * (U_2 + U_1)$$

$$X_3 = W * U \quad Y_3 = T * (U_2 * U_0 - W) - T_2 * U_3 \quad Z_3 = U_3 * V$$

  - Cost : **14M + 6A**

- For a twisted Edwards curve $T : ax^2 + y^2 = 1 + dx^2y^2$
  - Let $(x_1, y_1) \in T(\mathbb{F}_p)$ and $(x_2, y_2) \in T(\mathbb{F}_p)$
  - The addition of these two points is given by

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \qquad y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}$$

  - Using extended representation $(x_i, y_i) \to (X_i, Y_i, Z_i, T_i)$:

$$A = (Y_1 - X_1) * (Y_2 - X_2) \qquad B = (Y_1 + X_1) * (Y_2 + X_2) \qquad C = 2d * T_1 * T_2 \qquad D = 2Z_1$$

$$E = B - A \qquad F = D - C \qquad G = D + C \qquad H = B + A$$

$$X_3 = E * F \qquad Y_3 = G * H \qquad T_3 = E * H \qquad Z_3 = F * G$$

  - Cost : **8M + 8A**[6]

[6] Hisil, H., Wong, K. K. H., Carter, G., & Dawson, E. (2008). Twisted Edwards curves revisited. In *14th International Conference on the Theory and Application of Cryptology and Information Security*, Melbourne, Australia, December 7-11, 2008. (pp. 326-343). Springer Berlin Heidelberg.

Point accumulation in Hardware

Short Weierstrass	Mont.	Twisted Edwards	Scaled Edwards	Projective Extended Edwards			Scaled Edwards	Twisted Edwards	Montgomery	Short Weierstrass
a,b,α,s	$A = 3\alpha s$ $B = s$	$a = (A+2)/B$ $d = (A-2)/B$	$a' = -1$ $d' = -d/a$ $i = sqrt(-a)$				$a' = -1$ $d' = -d/a$	$a = -d/d'$ $d = -d'/a$	$A = 2(a+d)/(a-d)$ $B = 4(a-d)$	$a = (3-A^2)/3B^2$ $b = (2A^3-9A)/27B^3$
P_x	$P_x=s(x-\alpha)$	$P_x=P_x/P_y$	$P_x=iP_x$	P_x	$A=(P_y-P_x)*(Q_y-Q_x)$	$R_x=(B-A)*(P_z+P_z-C)$	$P_x=R_x/R_z$	$P_x=P_x/i$	$P_x=(1+P_y)/(1-P_y)$	$P_x=P_x/B+A/3B$
P_y	$P_y=sP_y$	$P_y=(P_x-1)/(P_x+1)$	P_y	P_y	$B=(P_y+P_x)*(Q_y+Q_x)$	$R_y=(B+A)*(P_z+P_z+C)$	$P_y=R_y/R_z$	P_y	$P_y=(1+P_y)/(P_x(1-P_y))$	$P_y=P_y/B$
				$P_t = P_xP_y$	$C=P_t*Q_t$	$R_t=(B-A)*(B+A)$				
				$P_z = 1$		$R_z=(P_z+P_z-C)*(P_z+P_z+C)$				
Q_x	$Q_x=s(x-\alpha)$	$Q_x=Q_x/Q_y$	$Q_x=iQ_x$	Q_x						
Q_y	$Q_y=sQ_y$	$Q_y=(Q_x-1)/(Q_x+1)$	Q_y	Q_y						
				$Q_t=d'Q_xQ_y$						

- For a twisted Edwards curve $T : ax^2 + y^2 = 1 + dx^2y^2$
  - With some pre/post-computation $(X_i, Y_i, Z_i, T_i) \rightarrow ((Y_i - X_i)/2, (Y_i + X_i)/2, Z_i, 4d * T_i)$:

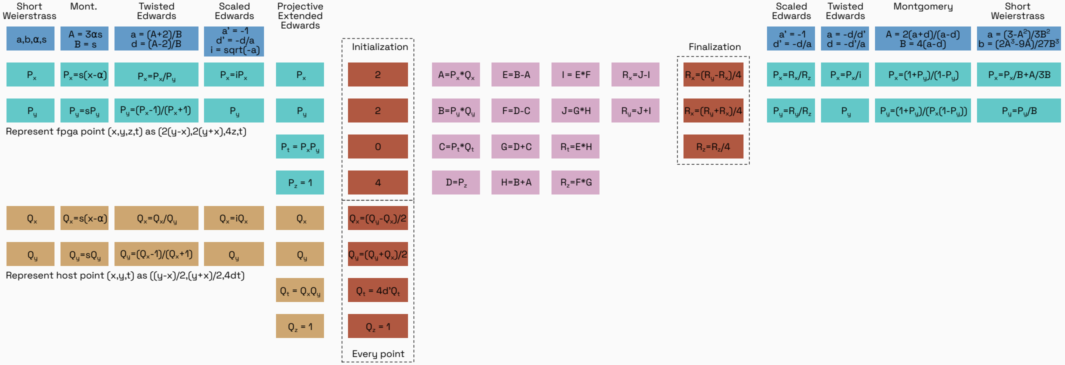
$$A = X_1 * X_2 \qquad B = Y_1 * Y_2 \qquad C = T_1 * T_2 \qquad D = Z_1$$

$$E = B - A \qquad F = D - C \qquad G = D + C \qquad H = B + A \qquad I = E * F \qquad J = G * H$$

$$X_3 = J - 1 \qquad Y_3 = J + 1 \qquad T_3 = E * H \qquad Z_3 = F * G$$

  - Cost : **7M + 6A**

Point accumulation in Hardware



	CycloneMSM[7]	Hardcaml[8]	PipeMSM[9]
Curve	BLS12-377	BLS12-377	BLS12-377
Representation	Montgomery $R = 2^{384}$		
Multiplier	3-layer Karatsuba	4-layer Karatsuba,	3-layer Karatsuba
Reduction	Montgomery	Barret	Barret
$P + Q$ latency	96 cycles	200 cycles	115 cycles
Frequency	250 MHz	278 MHz	125 MHz

[7] Aasaraai, K., Beaver, D., Cesena, E., Maganti, R., Stalder, N., & Varela, J. (2022). FPGA Acceleration of Multi-Scalar Multiplication: CycloneMSM. Cryptology ePrint Archive.

[8] https://zprize.hardcaml.com/

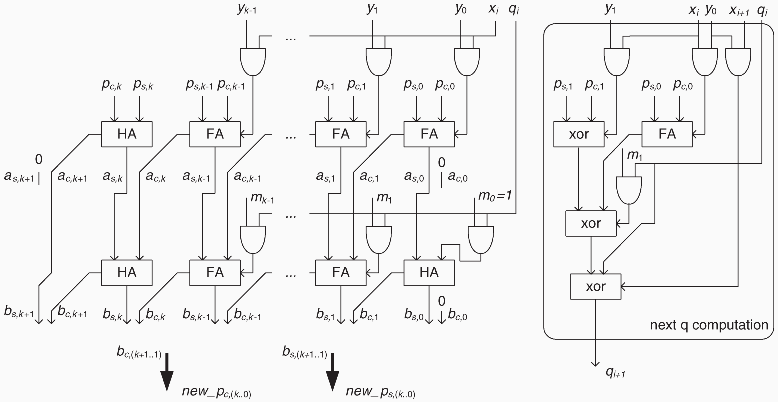
[9] Xavier, C. F. (2022). PipeMSM: Hardware acceleration for multi-scalar multiplication. Cryptology ePrint Archive.

Short Weierstrass	Mont.	Twisted Edwards	Scaled Edwards	Projective Extended Edwards	Initialization					Finalization	Scaled Edwards	Twisted Edwards	Montgomery	Short Weierstrass
a,b,α,s	A = 3αs B = s	a = (A+2)/B d = (A-2)/B	a' = -1 d' = -d/a i = sqrt(-a)								a' = -1 d' = -d/a	a = -d/d' d = -d'/a	A = 2(a+d)/(a-d) B = 4(a-d)	$a = (3-A^2)/3B^2$ $b = (2A^3-9A)/27B^3$
P_x	$P_x=s(x-α)$	$P_x=P_x/P_y$	$P_x=iP_x$	P_x	2^{k+1}	$A=P_x*Q_x$	E=B-A	I = E*F	$R_x=J-I$	$R_x=(R_y-R_x)/2^{k+2}$	$P_x=R_x/R_z$	$P_x=P_x/i$	$P_x=(1+P_y)/(1-P_y)$	$P_x=P_x/B+A/3B$
P_y	$P_y=sP_y$	$P_y=(P_x-1)/(P_x+1)$	P_y	P_y	2^{k+1}	$B=P_y*Q_y$	F=D-C	J=G*H	$R_y=J+I$	$R_x=(R_y+R_x)/2^{k+2}$	$P_y=R_y/R_z$	P_y	$P_y=(1+P_y)/(P_x(1-P_y))$	$P_y=P_y/B$
Represent fpga point (x,y,z,t) as (2(y-x),2(y+x),4z,t)														
				$P_t = P_xP_y$	0	$C=P_t*Q_t$	G=D+C	$R_t=E*H$		$R_z=R_z/2^{k+2}$				
				$P_z = 1$	2^{k+2}	$D=P_z$	H=B+A	$R_z=F*G$						
Q_x	$Q_x=s(x-α)$	$Q_x=Q_x/Q_y$	$Q_x=iQ_x$	Q_x	$Q_x=2^{k-1}(Q_y-Q_x)$									
Q_y	$Q_y=sQ_y$	$Q_y=(Q_x-1)/(Q_x+1)$	Q_y	Q_y	$Q_y=2^{k-1}(Q_y+Q_x)$									
Represent host point (x,y,t) as ((y-x)/2,(y+x)/2,4dt)														
				$Q_t = Q_xQ_y$	$Q_t = 2^{k+1}d'Q_t$									
				$Q_z = 1$	$Q_z = 2^k$									
					Every point									

Modular multiplication

1. Use Montgomery representation with $k = 384$
2. Decompose the Montgomery product into a cell array[10]
3. Combine with carry-save addition
4. Perform a Montgomery product in 96 cycles
5. Target 333MHz (technology bound)

[10] Sutter, G. D., Deschamps, J. P., & Imaña, J. L. (2010). Modular multiplication and exponentiation architectures for fast RSA cryptosystem based on digit serial computation. *IEEE Transactions on Industrial Electronics*, 58(7), 3101-3109.



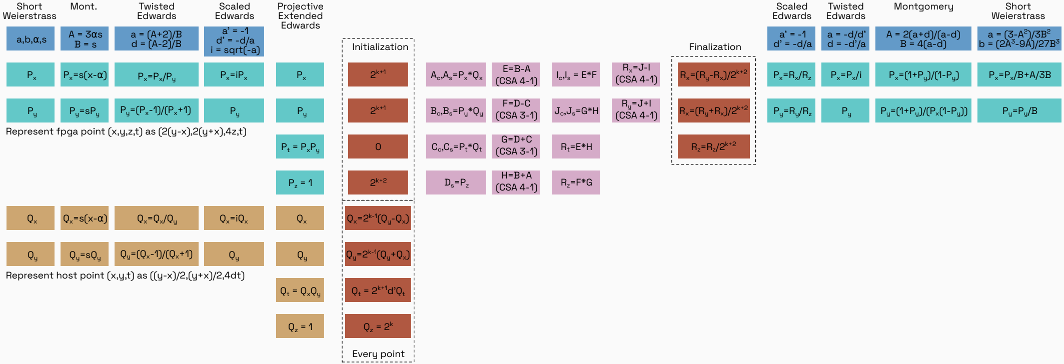
11

Sutter, G. D., Deschamps, J. P., & Imaña, J. L. (2010). Modular multiplication and exponentiation architectures for fast RSA cryptosystem based on digit serial computation. *IEEE Transactions on Industrial Electronics*, 58(7), 3101-3109.[11]

PARAMETER	Ours[12]			Hardcaml	CycloneMSM
Word size	4	6	8	13	48
LUTs	7399	10847	14411	29161	43445
Use of the FPGA (%)	0.63	0.92	1.22	2.47	3.67
FFs	3151	3159	3172	53948	48361
Use of the FPGA (%)	0.13	0.13	0.13	2.28	2.05
DSPs	0	0	0	428	324
Use of the FPGA (%)	0	0	0	6.26	4.74
Target period (ns)	3	4	5	3.6	4
WNS (ns)	0.078	0.017	0.126	0.261	0.038
FMAX (MHz)	342	251	205	299	252
Latency (Cycles)	100	68	50	76	40
Throughput (Mbps)	1313	1417	1574	1511	2419

[12] Implemented for the AMD Virtex UltraScale+ FPGA available in the Amazon EC2 F1 Instances.

Considering carry-save arithmetic



A general overview

ZERO-KNOWLEDGE

Protocol type:
not interactive

Type:
Pairing based
cryptography

Class:
SNARK

Relevant operation:
Multi Scalar
Multiplication

ECC

MSM algorithm:
Pippenger
Relevant operation:
Point addition

Elliptic curve:
BLS12-377
Point representation:
R/W

FIELD ARITHMETIC

Operand representation:
Montgomery
Multiplicaiton:
Montgomery cell array

Thanks !