NEON SIKE: Supersingular Isogeny Key Encapsulation on ARMv7

Amir Jalali[1], Reza Azarderakhsh[1], and Mehran Mozaffari Kermani[2]

Department of Computer and Electrical Engineering and Computer Science
Florida Atlantic University

Department of Computer Science and Engineering, University of South Florida

SPACE 2018

Overview

Plan

Quantum Computers

- Quantum Computers will be able to solve many problems in different areas:
  - Weather prediction
  - Medical and healthcare
  - Machine learning
  - Many other hard problems can be solved in polynomial-time

Quantum Computers

- Quantum Computers will be able to solve many problems in different areas:
  - Weather prediction
  - Medical and healthcare
  - Machine learning
  - Many other hard problems can be solved in polynomial-time
- Current PKC is also constructed on hard problems!
  - RSA: Discrete Logarithm Problem (DLP)
  - ECC: Elliptic Curve Discrete Logarithm Problem (ECDLP)
  - Shor's quantum algorithm can solve these problems in polynomial-time

Classical ECC

- Classical ECC is constructed on a single curve over a **large** prime p (256-bit).
- Based on elliptic curve group law, point multiplication is constructed.
- Computing $Q = [k]P$ is easy and fast.
- Given two points $P \in E(\mathbb{F}_p)$ and $Q = [k]P \in E(\mathbb{F}_p)$, finding $[k] \rightarrow O(p^{1/2})$

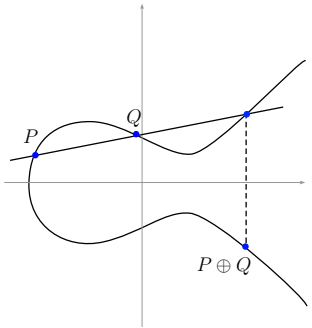


Figure: Geometrically representation of elliptic curves group law over $\mathbb{R}$.

Plan

Isogeny-based Cryptography

- Isogeny-based cryptography is constructed on a set of curves.
- Given two curves E and $E' = \phi(E)$, find ϕ ?

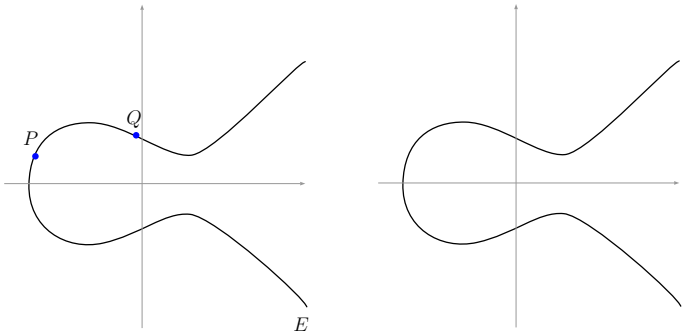


Figure: Isogeny maps

Isogeny-based Cryptography

- Isogeny-based cryptography is constructed on a set of curves.
- Given two curves E and $E' = \phi(E)$, find ϕ ?

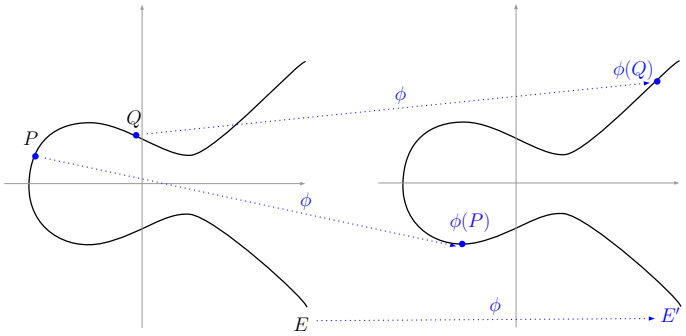


Figure: Isogeny maps

Isogenies of Elliptic Curves

Isogeny Kernel

Kernel of isogeny ϕ on a curve E, is a finite subgroup of points on E.

Isogenies of Elliptic Curves

Isogeny Kernel

Kernel of isogeny ϕ on a curve E, is a finite subgroup of points on E.

Isogeny

An isogeny ϕ is a group isomorphism for elliptic curves which has a finite kernel. Given a finite subgroup $G \in E_1$ there is a unique separable isogeny $\phi_G : E_1 \rightarrow E_2$ with kernel G.

Isogenies of Elliptic Curves

Isogeny Kernel

Kernel of isogeny ϕ on a curve E, is a finite subgroup of points on E.

Isogeny

An isogeny ϕ is a group isomorphism for elliptic curves which has a finite kernel. Given a finite subgroup $G \in E_1$ there is a unique separable isogeny $\phi_G : E_1 \to E_2$ with kernel G.

- The degree of isogeny $\deg(\phi) = \#\ker(\phi)$.
- For instance, if $G = \{-P, \mathcal{O}, P\}$, then $\deg(\phi_G) = 3$.

Isogenies of Elliptic Curves

Isogeny Kernel

Kernel of isogeny ϕ on a curve E, is a finite subgroup of points on E.

Isogeny

An isogeny ϕ is a group isomorphism for elliptic curves which has a finite kernel. Given a finite subgroup $G \in E_1$ there is a unique separable isogeny $\phi_G : E_1 \to E_2$ with kernel G.

- The degree of isogeny $\deg(\phi) = \#\ker(\phi)$.
- For instance, if $G = \{-P, \mathcal{O}, P\}$, then $\deg(\phi_G) = 3$.

Small Degree Isogeny Computation: Vélu's formula

Input: A generator of the kernel G (e.g., P) of the small degree isogeny.
Output: The image of E_1 (i.e., E_2) and the rational map to compute the point images.

Supersingular Isogeny-Based Cryptography Underlying Problem

- Consider two supersingular elliptic curves defined over a large prime extension field: $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, where p is a large prime.

Supersingular Isogeny-Based Cryptography Underlying Problem

- Consider two supersingular elliptic curves defined over a large prime extension field: $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, where p is a large prime.

Computational Supersingular Isogeny (CSSI) Problem

Given $P, Q \in E_1$ and $\phi(P), \phi(Q) \in E_2$, retrieve the secret isogeny map ϕ

Supersingular Isogeny-Based Cryptography Underlying Problem

- Consider two supersingular elliptic curves defined over a large prime extension field: $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, where p is a large prime.

Computational Supersingular Isogeny (CSSI) Problem

Given $P, Q \in E_1$ and $\phi(P), \phi(Q) \in E_2$, retrieve the secret isogeny map ϕ

- The best known quantum attack is based on **Claw's finding algorithm** $\rightarrow O(\ell^{1/3})$
- Claw finding algorithm complexity for SIKE and SIDH:
  - $O(p^{1/6})$ $\rightarrow$Quantum attacks
- The best known classical attack is based on **meet in the middle** $\rightarrow O(\ell^{1/2})$
  - $O(p^{1/4})$ $\rightarrow$Classical attacks

Plan

SIKE

- SIKE: Supersingular Isogeny Key Encapsulation.

SIKE

- SIKE: Supersingular Isogeny Key Encapsulation.
- One of the submissions to the NIST PQC standardization.

SIKE

- SIKE: Supersingular Isogeny Key Encapsulation.
- One of the submissions to the NIST PQC standardization.
- Based on Jao-De Feo PKE scheme, but it is IND-CCA secure $\rightarrow$ Static keys

SIKE

- SIKE: Supersingular Isogeny Key Encapsulation.
- One of the submissions to the NIST PQC standardization.
- Based on Jao-De Feo PKE scheme, but it is IND-CCA secure $\rightarrow$ Static keys

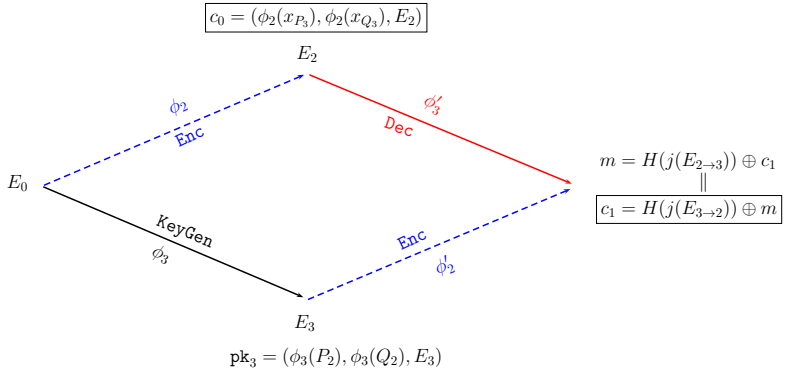


Figure: Isogeny-based public key encryption.

SIKE

Bob		Alice
KeyGen:		
$\mathtt{pk}_B = [E_B, \phi_B(x_{P_A}), \phi_B(x_{Q_A})]$		
$s \in_R \{0,1\}^\ell$		
		Encapsulation:
		$m \in_R \{0,1\}^\ell$
		$r = G(\mathtt{pk}_B, m)$
		$\mathtt{pk}_A(r) = [E_A, \phi_A(x_{P_B}), \phi_A(x_{Q_B})]$
		$j_{\mathtt{inv}} = j(E_{AB})$
		$\mathtt{Enc}(\mathtt{pk}_B, m, r) \rightarrow (c_0, c_1)$
		$(c_0, c_1) = (\mathtt{pk}_A(r), G(j_{inv}) \oplus m)$
		$K = H(m \parallel (c_0, c_1))$
	$\xleftarrow{(c_0,c_1)}$	
Decapsulation:		
$j_{\mathtt{inv}} = j(E_{BA})$		
$m' = c_1 \oplus G(j_{\mathtt{inv}})$		
$r' = G(\mathtt{pk}_B, m')$		
If $\mathtt{pk}_A(r') = c_0 \rightarrow K = H(m' \parallel (c_0, c_1))$		
If $\mathtt{pk}_A(r') \neq c_0 \rightarrow K = H(s \parallel (c_0, c_1))$		

Figure: SIKE protocol.

SIKE Implementation Parameters

Table: SIKE parameters

NIST level	$p = 2^{e_2}3^{e_3} - 1$	Length (bits)	$\min(\sqrt[3]{2^{e_2}}, \sqrt[3]{3^{e_3}})$	Quantum Security
3 (SIKEp503)	$2^{250}3^{159} - 1$	503	1.26×2^{83}	83
5 (SIKEp751)	$2^{372}3^{239} - 1$	751	1.00×2^{124}	124
7 (SIKEp964)	$2^{486}3^{301} - 1$	964	1.02×2^{159}	159

- Smallest key and ciphertext size compared to other candidates.

Table: SIKE vs. Lattice-based KEMs on Intel processors

Scheme	Primitive	PQ-Security	Encaps+Decaps (ms)	Size of Encap. (KB)
NTRU-KEM	NTRU	123	0.03	1.3
Kyber	M-LWE	161	0.07	1.2
FrodoKEM	LWE	103-150	1.2-2.3	9.5-15.4
SIKE	SI	84-125	10-30	0.4-0.6

Plan

NEON Instruction Set for Field Arithmetic Implementation

- ARMv7-A processors $\rightarrow$ 32-bit general registers and 128-bit NEON vectors.

NEON Instruction Set for Field Arithmetic Implementation

- ARMv7-A processors $\rightarrow$ 32-bit general registers and 128-bit NEON vectors.
- Exploit SIMD capabilities of ARMv7-A cores for arithmetic implementation

NEON Instruction Set for Field Arithmetic Implementation

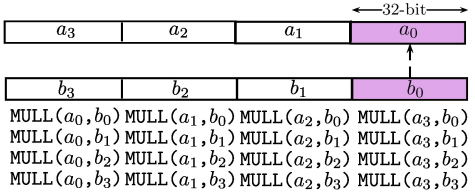
- ARMv7-A processors $\rightarrow$ 32-bit general registers and 128-bit NEON vectors.
- Exploit SIMD capabilities of ARMv7-A cores for arithmetic implementation
  - NEON Assembly Implementation:

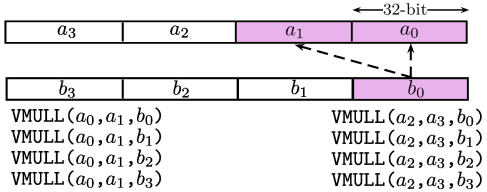
NEON Instruction Set for Field Arithmetic Implementation

- ARMv7-A processors $\rightarrow$ 32-bit general registers and 128-bit NEON vectors.
- Exploit SIMD capabilities of ARMv7-A cores for arithmetic implementation
  - NEON Assembly Implementation:
    - SIMD instructions reduce the total number of multiplications significantly

NEON Instruction Set for Field Arithmetic Implementation

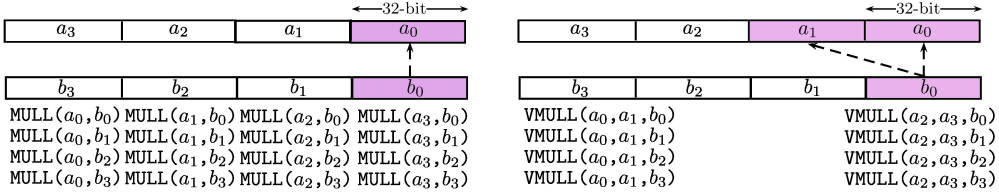
- ARMv7-A processors $\rightarrow$ 32-bit general registers and 128-bit NEON vectors.
- Exploit SIMD capabilities of ARMv7-A cores for arithmetic implementation
  - NEON Assembly Implementation:
    - SIMD instructions reduce the total number of multiplications significantly
    - 128 $\times$ 128-bit multiplication using A32 and NEON:





NEON Instruction Set for Field Arithmetic Implementation

- ARMv7-A processors $\rightarrow$ 32-bit general registers and 128-bit NEON vectors.
- Exploit SIMD capabilities of ARMv7-A cores for arithmetic implementation
  - NEON Assembly Implementation:
    - SIMD instructions reduce the total number of multiplications significantly
    - 128 $\times$ 128-bit multiplication using A32 and NEON:



- 16 $\times$ `MULL` instructions in A32 vs. 8 $\times$ `VMULL` in NEON Vector Instructions.

Additive Karatsuba Multiplication

- Additive Karatsuba multiplication

$$a \times b = A_1 \cdot B_1 \cdot 2^n + [(A_1 + A_0)(B_1 + B_0) - A_1 \cdot B_1 - A_0 \cdot B_0] \cdot 2^{\frac{n}{2}} + A_0 \cdot B_0.$$

Additive Karatsuba Multiplication

- Additive Karatsuba multiplication

$$a \times b = A_1 \cdot B_1 \cdot 2^n + [(A_1 + A_0)(B_1 + B_0) - A_1 \cdot B_1 - A_0 \cdot B_0] \cdot 2^{\frac{n}{2}} + A_0 \cdot B_0.$$

- Two-level Karatsuba for `SIKEp964` implementation:

$$A_1 B_1 = A_{11} \cdot B_{11} \cdot 2^{\frac{n}{2}} + [(A_{11} + A_{10})(B_{11} + B_{10}) - A_{11} \cdot B_{11} - A_{10} \cdot B_{10}] \cdot 2^{\frac{n}{4}} + A_{10} \cdot B_{10}$$

$$A_0 B_0 = A_{01} \cdot B_{01} \cdot 2^{\frac{n}{2}} + [(A_{01} + A_{00})(B_{01} + B_{00}) - A_{01} \cdot B_{01} - A_{00} \cdot B_{00}] \cdot 2^{\frac{n}{4}} + A_{00} \cdot B_{00}.$$

Additive Karatsuba Multiplication

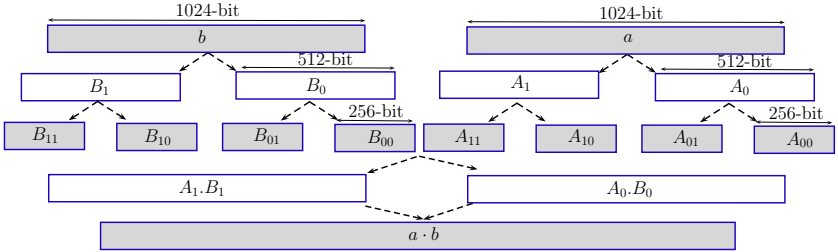
- Additive Karatsuba multiplication

$$a \times b = A_1 \cdot B_1 \cdot 2^n + [(A_1 + A_0)(B_1 + B_0) - A_1 \cdot B_1 - A_0 \cdot B_0] \cdot 2^{\frac{n}{2}} + A_0 \cdot B_0.$$

- Two-level Karatsuba for `SIKEp964` implementation:

$$A_1 B_1 = A_{11} \cdot B_{11} \cdot 2^{\frac{n}{2}} + [(A_{11} + A_{10})(B_{11} + B_{10}) - A_{11} \cdot B_{11} - A_{10} \cdot B_{10}] \cdot 2^{\frac{n}{4}} + A_{10} \cdot B_{10}$$

$$A_0 B_0 = A_{01} \cdot B_{01} \cdot 2^{\frac{n}{2}} + [(A_{01} + A_{00})(B_{01} + B_{00}) - A_{01} \cdot B_{01} - A_{00} \cdot B_{00}] \cdot 2^{\frac{n}{4}} + A_{00} \cdot B_{00}.$$



Tailored NEON-based Montgomery Reduction

Costello-Longa-Naehrig (2016):

- Faster Mont. reduction can be computed over $\hat{p} = p + 1$.
- $\hat{p}$ has multiple words equal to ”0” which can be ignored in the computations.
- Tailored Mont. reduction for each prime in NEON.

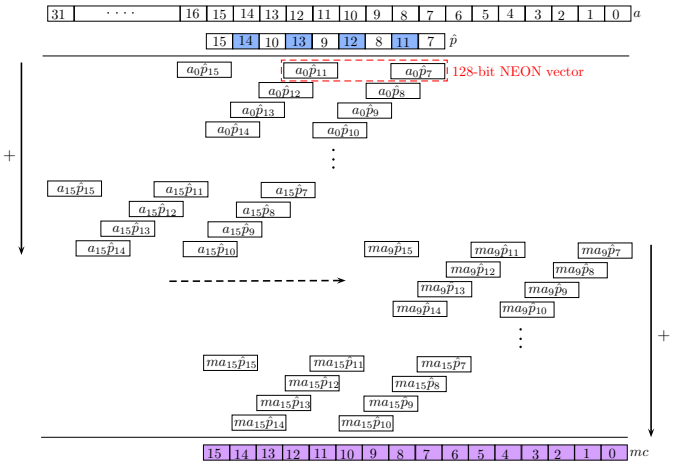


Figure: Mont. reduction over `SIKEp503`

Plan

Performance

- Finite field arithmetic → NEON assembly.
- Protocol level implementation → SIKE reference implementation in C.
- Target processors:
  - BeagleBone development board → Cortex-A8 (1.0 GHz).
  - NVIDIA Jetson-TK1 → Cortex-A15 (2.3 GHz).

Scheme	Operation	**NEON ASM**		Optimized C	
		Cortex-A8	Cortex-A15	Cortex-A8	Cortex-A15
SIKEp503	KeyGen.	99	68	813	577
	Encap.	162	112	1,339	910
	Decap.	174	121	1,424	955
	Total	**435**	**301**	3,576	2,442
SIKEp751	KeyGen.	364	280	2,842	2,089
	Encap.	589	439	4,598	3,331
	Decap.	618	491	4,944	3,531
	Total	**1,571**	**1,210**	12,384	8,951
SIKEp964	KeyGen.	870	635	6,037	4,409
	Encap.	1,504	1,098	10,376	7,678
	Decap.	1,598	1,176	10,835	7,963
	Total	**3,972**	**2,909**	27,248	20,050

Plan

Conclusion and Future Work

- An optimized implementation of SIKE on ARMv7-A processors is presented.
- Taking advantage of NEON instructions and the special shape of SIKE primes:
  - Optimized field multiplication.
  - Optimized Montgomery reduction.
- Benchmarks show almost 7 times performance improvement compared to reference C implementations.
- SIKE performance is challenging on low-power embedded devices.
- Our initial evaluation shows a single KEM can take upto **several seconds** on power-efficient Cortex-M processors.
- We plan to provide an optimized library on 16-bit power-efficient cores in the future.

Thank You!