NEON-SIDH: Efficient Implementation of Supersingular Isogeny Diffie-Hellman Key Exchange Protocol on ARM

Paper by: Brian Koziel (Texas Instruments) [corresponding author, kozielbrian@gmail.com], Amir Jalali (Rochester Institute of Technology), Reza Azarderakhsh (Florida Atlantic University), David Jao (University of Waterloo), and Mehran Mozaffari-Kermani (Rochester Institute of Technology)

CANS 2016
Milan, Italy

Outline

Introduction

- Supersingular isogeny Diffie-Hellman (SIDH) as a strong quantum-resistant cryptographic primitive for NIST's PQC standardization
  - Originally presented by Jao and De Feo at PQCrypto 2011
  - Provides small keys, forward secrecy and a Diffie-Hellman key exchange
  - Can be visualized as moving from elliptic curve to elliptic curve
- This work analyzes SIDH implementation on ARMv7 cores targeted at embedded processors
  - We show that affine isogeny formulas are still useful for ARMv7

Contributions

- We provide efficient libraries for SIDH using highly optimized C and ASM.
- We present fast and secure prime candidates for 85-bit, 128-bit, and 170-bit quantum security levels.
- We provide hand-optimized finite field arithmetic computations over various ARM-powered processors to produce constant-time arithmetic that is 3 times as fast as GMP.
- We analyze the effectiveness of projective and affine isogeny computation schemes.
- We provide implementation results for embedded devices running Cortex-A8 and Cortex-A15.

SIDH Overview

- Proposed by David Jao and Luca De Feo[1]
- Public Parameters
  - Smooth Isogeny Prime - $p = \ell_A^{e_A}\ell_B^{e_B} f \pm 1$, where ℓ_A and ℓ_B are small primes, e_A and e_B are positive integers, and f is a small cofactor to make the number prime
  - Starting Supersingular Elliptic Curve, $E_0/\mathbb{F}_{p^2}$
  - Torsion bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ over $E_0[\ell_A^{e_A}]$ and $E_0[\ell_B^{e_B}]$, respectively
- Classical and quantum security is approximately $\sqrt[4]{p}$ and $\sqrt[6]{p}$, respectively.
  - Based on the difficulty of computing isogenies between supersingular elliptic curves

[1] Jao, D., De Feo, L.: *Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies*. PQCrypto 2011: 19-34.

SIDH Overview

- Each round is broken into computing a double point multiplication, $R = mP + nQ$, where m and n are secret scalars, and using R as a secret kernel for an isogeny, $\phi : E \to E/\langle R \rangle$.
  - $\phi_A : E \to E/\langle m_A P_A + n_A Q_A \rangle = E_A$ for Alice and $\phi_B : E \to E/\langle m_B P_B + n_B Q_B \rangle = E_B$ for Bob
- After the first round, Alice sends $\{E_A, \phi_A(P_B), \phi_A(Q_B)\}$ and Bob sends $\{E_B, \phi_B(P_A), \phi_B(Q_A)\}$
- After the second round, Alice and Bob have isomorphic curves, so the j-invariant can be used as a shared secret key.
  - $\phi'_A : E_B \to E_B/\langle m_A \phi_B(P_A) + n_A \phi_B(Q_A) \rangle = E_{AB}$ for Alice and $\phi'_B : E_A \to E_A/\langle m_B \phi_A(P_B) + n_B \phi_A(Q_B) \rangle = E_{BA}$ for Bob
  - $j(E_{AB}) = j(E_{BA})$

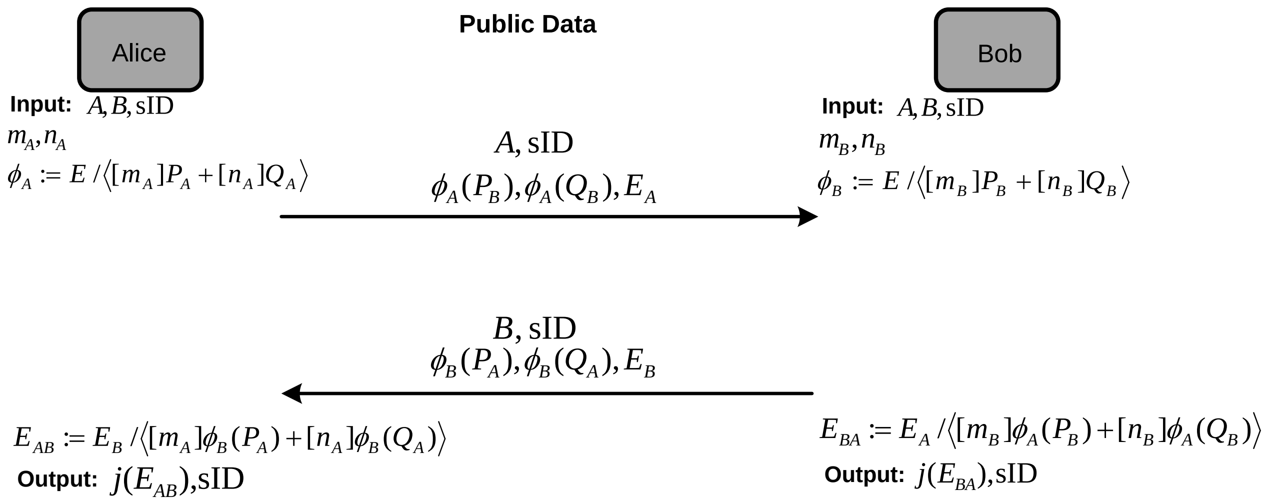


Figure: SIDH Protocol

SIDH Computations

- Goal: Optimize double point multiplication and large-degree isogeny
- Solution: All arithmetic is performed on Kummer line of Montgomery curve
  - Represent points as $(x, y) \rightarrow (X : Z)$, where $x = X/Z$. P and $-P$ produce same isogenies
- 3-point Montgomery differential ladder for double point multiplication if $m = 1$[1]
  - Computes $P + [t]Q$ at each iteration
- Fast Montgomery arithmetic to compute isogenies of degree 2, 3, and 4
  - Affine formulas that require an inversion for each isogeny computation[1]
  - Projective formulas that require an inversion at the end of the round[2]

[1] De Feo, L., Jao, D., Plut, J.: *Towards Quantum-resistant Cryptosystems from Supersingular Elliptic Curve Isogenies*. Journal of Mathematical Cryptography, 2014, 8(3):209-247.
[2] Costello, C., Longa, P., Naehrig, M.: *Efficient Algorithms for Supersingular Isogeny Diffie-Hellman*. CRYPTO 2016: 572-601

SIDH-Friendly Primes

- Smooth Isogeny Prime - $p = \ell_A^{e_A} \ell_B^{e_B} \cdot f \pm 1$, where ℓ_A and ℓ_B are small primes, e_A and e_B are positive integers, and f is a small cofactor to make the number prime
- Fast known point multiplications and isogeny formulas for $\ell_A = 2$ and $\ell_B = 3$
- Security of a large-degree isogeny is $\sqrt[3]{\ell^e}$
  - Quantum claw finding problem

SIDH-Friendly Primes

- Find several different primes at each security level for a variety of optimizations
  - Such as redundant radix representations, lazy reduction, etc.
- Prime search criteria:
  - **Security**: The relative security of SIDH over a prime is based on $\min(\ell_A^a, \ell_B^b)$.
  - **Size**: These primes should feature a size slightly less than a power of 2 to allow for some speed optimizations such as lazy reduction and carry cancelling, while still featuring a high quantum security.
  - **Speed**: These primes efficiently use space to reduce the number of operations per field arithmetic, but also have nice properties for the field arithmetic. Notably, all primes of the form $p = 2^a \ell_B^b \cdot f - 1$ will have the Montgomery friendly property because the least significant half of the prime will have all bits set to '1'.

Proposed SIDH-Friendly Primes

Security Level	Prime Size (bits)	$p = \ell_A^{e_A}\ell_B^{e_B} \cdot f \pm 1$	$\min(\ell_A^{e_A}, \ell_B^{e_B})$	Classical Security	Quantum Security
p_{512}	499	$2^{251}3^{155}5 - 1$	3^{155}	123	82
	503	$2^{250}3^{159} - 1$	2^{250}	125	83
	510	$2^{252}3^{159}37 - 1$	2^{252}	126	84
p_{768}	751	$2^{372}3^{239} - 1$	2^{372}	186	124
	758	$2^{378}3^{237}17 - 1$	3^{237}	188	125
	766	$2^{382}3^{238}79 - 1$	3^{238}	189	126
p_{1024}	980	$2^{493}3^{307} - 1$	3^{307}	243	162
	1004	$2^{499}3^{315}49 - 1$	2^{499}	249	166
	1008	$2^{501}3^{316}41 - 1$	3^{316}	250	167
	1019	$2^{508}3^{319}35 - 1$	3^{319}	253	168

ARMv7 Finite-Field Arithmetic

- Since supersingular curves can be defined over $\mathbb{F}_{p^2}$, all finite-field operations are over $\mathbb{F}_{p^2}$
  - Optimize at $\mathbb{F}_p$ and join operations to get $\mathbb{F}_{p^2}$
- With choice of $\ell_A = 2$, -1 is not a quadratic residue and $x^2 + 1$ is an efficient modulus
- The following arithmetic utilizes a non-redundant scheme for registers

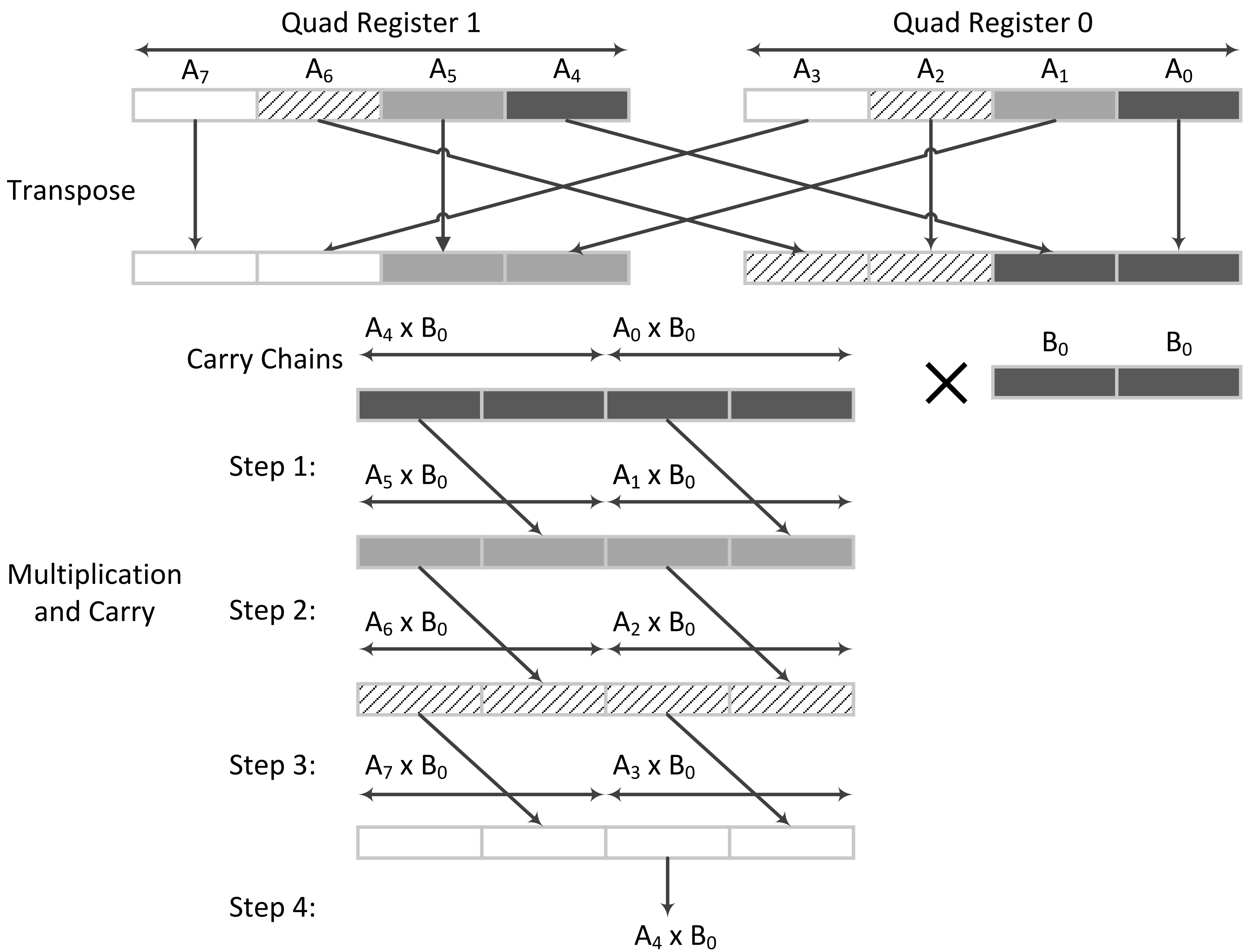
Finite-Field Addition

- $A + B = C$, where $A, B, C \in \mathbb{F}_p$
  - If $C \geq p$, then $C = C - p$
- Use *ldmia* and *stmia* instructions to load multiple registers at a time and iteratively add with carry
- For conditional subtraction, perform a masked subtraction
  - $C = C - \{0 \text{ if } C < p, p \text{ if } C \geq p\}$

Finite-Field Multiplication

- $A \times B = C$, where $A, B, C \in \mathbb{F}_p$
- Requires a reduction from $2m$ bits to m bits, so Montgomery reduction was used
- Perform separated multiply and reduce with Cascade Operand Scanning (COS) method[1]
  - Utilizes ARM-NEON vector unit
  - Efficiently performs many 32×256 bit multiplications by utilizing a transpose of inputs to minimize data dependencies and expands to 512×512 bits
  - With choice of primes, we reduce the complexity from $k^2 + k$ to k^2 single-precision multiplications, where k is the number of words in the field

Finite-Field Multiplication



Finite-Field Multiplication and Squaring

- Base multiplier performs 512-bit multiplications
- Karatsuba’s method is used to perform 1024-bit multiplications
- Squaring is similar to multiplication, but partial products can be reused
    - Approximately 75% of the cycles for a multiplication

Finite-Field Inversion

- Finds some A^{-1} such that $A \cdot A^{-1} = 1$, where $A, A^{-1} \in \mathbb{F}_p$
- Fermat's little theorem performs $A^{-1} = A^{p-2}$
  - Complexity $O(\log^3 n)$
- Extended Euclidean Algorithm (EEA) or Kaliski Montgomery Inverse
  - EEA finds $ax + by = \gcd(a, b)$ to perform inverse
  - Complexity $O(\log^2 n)$
- Choice of EEA for fast inversions with affine formulas
  - Timing attack countermeasure: Multiplying value to be inverted before and after by a random value
  - GNU Multiprecision Library (GMP) implements heavily optimized inversion

Extension Field Arithmetic

- Let $A = (A_0, A_1)$, $B = (B_0, B_1) \in \mathbb{F}_{p^2}$.The results of operations in $\mathbb{F}_{p^2}$ are $C = (C_0, C_1)$

$$
\begin{aligned}
A + B &= (A_0 + B_0, A_1 + B_1) \\
A - B &= (A_0 - B_0, A_1 - B_1) \\
A \times B &= \begin{array}{c}(A_0 B_0 - A_1 B_1, (A_0 + A_1), \\ (B_0 + B_1) - A_0 B_1 - A_1 B_0)\end{array} \\
A^2 &= ((A_0 + A_1)(A_0 - A_1), 2A_0 A_1) \\
A^{-1} &= (A_0 (A_0^2 + A_1^2)^{-1}, -A_1 (A_0^2 + A_1^2)^{-1})
\end{aligned}
$$

Affine or Projective Isogenies

- Here we compare the relative costs of affine and projective isogenies
- Let I, M, and S refer to inversion, multiplication, and squaring in $\mathbb{F}_p$, respectively. A tilde above the letter indicates that the operation is in $\mathbb{F}_{p^2}$.

Table: Affine isogeny formulas vs. projective isogenies formulas

Computation	Affine Cost	Projective Cost
Point Mult-by-3	$7\tilde{M}+4\tilde{S}$	$8\tilde{M}+5\tilde{S}$
Iso-3 Computation	$1\tilde{I}+5\tilde{M}+1\tilde{S}$	$3\tilde{M}+3\tilde{S}$
Iso-3 Evaluation	$4\tilde{M}+2\tilde{S}$	$6\tilde{M}+2\tilde{S}$
Point Mult-by-4	$6\tilde{M}+\tilde{S}$	$8\tilde{M}+4\tilde{S}$
Iso-4 Computation	$1\tilde{I}+3\tilde{M}$	$5\tilde{S}$
Iso-4 Evaluation	$6\tilde{M}+4\tilde{S}$	$9\tilde{M}+1\tilde{S}$

Affine or Projective Isogenies

Table: Comparison of break-even inversion/multiplication ratios for large-degree isogenies at different security levels. When the inversion over multiplication ratio is at the break-even point, affine isogenies require approximately the same cost as projective isogenies. Ratios smaller than these numbers are faster with affine formulas.

Prime	Alice R1 Iso	Bob R1 Iso	Alice R2 Iso	Bob R2 Iso
p_{512}	$\tilde{I} = 20.87\tilde{M}$	$\tilde{I} = 19.26\tilde{M}$	$\tilde{I} = 17.87\tilde{M}$	$\tilde{I} = 13.26\tilde{M}$
p_{768}	$\tilde{I} = 22.73\tilde{M}$	$\tilde{I} = 20.48\tilde{M}$	$\tilde{I} = 19.73\tilde{M}$	$\tilde{I} = 14.48\tilde{M}$
p_{1024}	$\tilde{I} = 23.41\tilde{M}$	$\tilde{I} = 21.15\tilde{M}$	$\tilde{I} = 20.41\tilde{M}$	$\tilde{I} = 15.15\tilde{M}$
p_{512}	$I = 52.62M$	$I = 47.78M$	$I = 43.62M$	$I = 29.78M$
p_{768}	$I = 58.20M$	$I = 51.44M$	$I = 49.20M$	$I = 33.46M$
p_{1024}	$I = 60.23M$	$I = 53.46M$	$I = 51.23M$	$I = 35.46M$

Affine or Projective Isogenies

- I/M ratio for ARM processors is generally much smaller than PC's
- From the last slide, the breakeven points for p_{512} ranges from 29.78 to 52.62
  - Thus, improvements in speed can be achieved from affine isogeny formulas

Table: Comparison of I/M ratios for various computer architectures based on GMP library

Architecture	Device	I/M ratio		
		p_{512}	p_{768}	p_{1024}
ARMv7 Cortex-A8	Beagle Board Black	7.0	6.4	6.1
ARMv7 Cortex-A15	Jetson TK1	7.1	6.1	5.9
ARMv8 Cortex-A53	Linaro HiKey	8.2	7.3	6.5
Haswell x86-64	i7-4790k	14.9	14.7	13.8

ARMv7 Results

- Benchmarked using BeagleBoard Black (Cortex-A8 @ 1.0 GHz) and Jetson TK1 (Cortex-A15 @ 2.3 GHz)
- GMP version 6.1.0
- Works with any valid parameters file

$$
\begin{aligned}
p_{512} &= 2^{250}3^{159} - 1 \\
p_{768} &= 2^{372}3^{239} - 1 \\
p_{1024} &= 2^{501}3^{316}41 - 1
\end{aligned}
$$

BeagleBoard Black Results

Table: Timing results of key exchange on Beagle Board Black ARMv7 device for different security levels

Beagle Board Black (ARM v7) Cortex-A8 at 1.0 GHz using C							
Field Size	$\mathbb{F}_p$ [cc]					Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	Alice	Bob
p_{512}	115	1866	2295	3429	40100	483,968	514,786
p_{768}	142	3652	4779	6325	71500	1,406,381	1,525,215
p_{1024}	168	5925	8202	10150	111900	3,135,526	3,367,448

Beagle Board Black (ARM v7) Cortex-A8 at 1.0 GHz using ASM and NEON							
Field Size	$\mathbb{F}_p$ [cc]					Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	Alice	Bob
p_{512}	70	718	953	962	40100	216,503	229,206
p_{1024}	120	2714	3723	3956	111900	1,597,504	1,708,383

Jetson TK1 Results

Table: Timing results of key exchange on NVIDIA Jetson TK-1 ARMv7 device for different security levels

Jetson TK-1 Board (ARM v7) Cortex-A15 at 2.3 GHz using C							
Field Size	$\mathbb{F}_p$ [cc]					Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	Alice	Bob
p_{512}	83	926	1152	2271	24302	285,026	302,332
p_{768}	99	1679	2403	4024	39100	783,303	848,461
p_{1024}	117	2955	4144	6053	59800	1,728,183	1,851,782

Jetson TK-1 Board (ARM v7) Cortex-A15 at 2.3 GHz using ASM and NEON							
Field Size	$\mathbb{F}_p$ [cc]					Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	Alice	Bob
p_{512}	39	516	640	732	24302	148,003	154,657
p_{1024}	73	1856	2464	2961	59800	1,118,644	1,140,626

Comparison of Results

Table: Comparison of affine and projective isogeny implementations on ARM Cortex-A15 embedded processors. Our work and Costello et al.'s was done on a Jetson TK1 and Azarderakhsh et al.'s was performed on an Arndale ARM Cortex-A15. Costello et al's implementation only supports generic arithmetic for ARM.

Work	Field Size [bits]	Iso. Eq.	Timings [$cc \times 10^6$]				
			Alice R1	Bob R1	Alice R2	Bob R2	Total
Costello et al.	751	Proj.	1,794	2,120	1,665	2,001	7,580
Azarderakhsh et al.	521	Affine	N/A	N/A	N/A	N/A	1,069
	771		N/A	N/A	N/A	N/A	3,009
	1035		N/A	N/A	N/A	N/A	6,477
This work	503	Affine	83	87	66	68	302
	751		437	474	346	375	1,632
	1008		603	657	516	484	2,259

Conclusions

- Efficient implementation of SIDH on ARMv7 platforms
- Proposed several fast SIDH-friendly primes
- Hand-optimized finite-field arithmetic $\rightarrow$ up to 3 times faster than GMP
- Analysis of the efficiency of affine and projective isogeny formulas $\rightarrow$ ARMv7 can benefit from affine
- Implementations on BeagleBoard Black and Jetson TK1 $\rightarrow$ currently fastest known implementations for ARMv7
- Push for robust and high-performance implementations for standardization of SIDH by NIST

Thank You!