Towards Optimized and Constant-Time CSIDH on Embedded Devices

Amir Jalali[1], Reza Azarderakhsh[1], Mehran Mozaffari Kermani[2], and David Jao[3]

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

Department of Computer Science and Engineering, University of South Florida

Department of Combinatorics and Optimization, University of Waterloo

COSADE 2019

Quantum Computers

- Current public-key cryptography is based on the following 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
- Post-quantum cryptography is based on hard problems that are hard even on a quantum computer:
  - Lattice-based cryptography
  - Code-based cryptography
  - Hash-based cryptography
  - Multivariate cryptography
  - Isogeny-based cryptography

Isogeny-based Cryptography

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

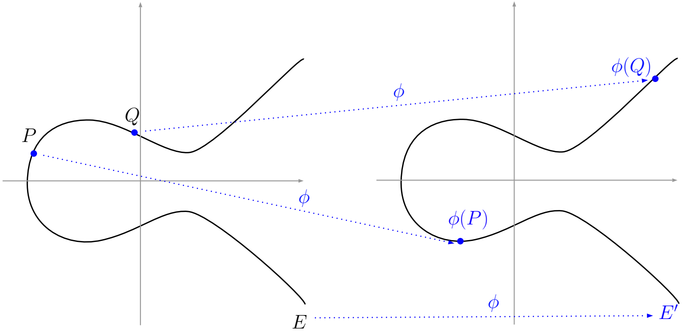

Figure: Isogeny maps

Isogenies of Elliptic Curves

Isogeny Kernel

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

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(ϕ).
- 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.

- Recently proposed Diffie-Hellman scheme on commutative group action.
- SIDH is defined over $E(\mathbb{F}_{p^2}) \rightarrow$ Not Commutative!
- CSIDH is defined over $E(\mathbb{F}_p) \rightarrow$ Commutative!
- Alice and Bob walk in two different isogeny graphs on the same isogeny class.

Alice		**Bob**
$\mathtt{SK}_A = (e_{A1}, \cdots, e_{An})$		$\mathtt{SK}_B = (e_{B1}, \cdots, e_{Bn})$
$[\mathfrak{a}] = [\mathfrak{l}_1^{e_{A1}} \cdots \mathfrak{l}_n^{e_{An}}]$		$[\mathfrak{b}] = [\mathfrak{l}_1^{e_{B1}} \cdots \mathfrak{l}_n^{e_{Bn}}]$
$\mathtt{PK}_A = [\mathfrak{a}]E_0 = E_A$		$\mathtt{PK}_B = [\mathfrak{b}]E_0 = E_B$
	$\xleftarrow{E_B}$	
	$\xrightarrow{E_A}$	
$\mathtt{Shared}_A = [\mathfrak{a}]E_B = [\mathfrak{a}][\mathfrak{b}]E_0$		$\mathtt{Shared}_B = [\mathfrak{b}]E_A = [\mathfrak{b}][\mathfrak{a}]E_0$

Figure: CSIDH key exchange.

CSIDH vs. SIDH

	CSIDH	SIDH
Speed (NIST level 1)	100 ms	10 ms
Public key size	64 bytes	330 bytes
Key compression	N/A	196 bytes
Constant-time	No	Yes
Best quantum attack	subexponential	$p^{1/6}$

Advantages and disadvantages of CSIDH:

- Key size is very small.
- Fast and straightforward **key validation**.
- Much slower and scales poorly against attacks.

This work: The evaluation of a constant-time CSIDH on embedded devices.

Related work

- Castryck et al. (`ia.cr/2018/383`) — original implementation
- Meyer and Reith (`ia.cr/2018/782`) — faster implementation with some constant-time ideas
- Meyer et al. (`ia.cr/2018/1198`) — claimed constant-time CSIDH
- Onuki et al. (`ia.cr/2019/353`) — claimed (faster) constant-time CSIDH

Is it really constant time?

> *"Our implementation allows variance the computational time with randomness that does not relate to secret information. Applying our method to an implementation based on a stricter definition of constant-time is a future work."*
> *—Onuki et al.*

Point Multiplication

- Compute $[k]P$ in constant-time to be side-channel attack resistant.
- Castryck et al. implementation: Fast, but totally vulnerable to DPA and SPA.
- This work: Constant-time variant of the Montgomery ladder:

Algorithm 1: Constant-time variable length scalar multiplication

Input : $k = \sum_{i=0}^{n-1} k_i 2^i$ and $\mathtt{x}(P)$ for $P \in E(\mathbb{F}_p)$.
Output: $(X_k, Z_k) \in \mathbb{F}_p^2$ s.t. $(X_k : Z_k) = \mathtt{x}([k]P)$.

1: $X_R \leftarrow X_P, Z_R \leftarrow Z_P$
2: $X_Q \leftarrow 1, Z_Q \leftarrow 0$
3: **for** $i = n-2$ **downto** 0 **do**
4: $\quad (Q, R) \leftarrow \mathtt{cswap}(Q, R, (k_i \ \mathtt{xor} \ k_{i+1}))$
5: $\quad (Q, R) \leftarrow \mathtt{xDBLADD}(Q, R, P)$
6: **end for**
7: $(Q, R) \leftarrow \mathtt{cswap}(Q, R, k_0)$
8: **return** Q

Variable-time Group Action

Algorithm 2: Variable-time secret key decoding (Castryck et al.)

for $i = 0$ **to** $n - 1$ **do**
  if $e_i > 0$ **then**
    $e_i(0) = e_i, e_i(1) = 0$
    $k(1) \leftarrow k(1) \cdot \ell_i$
  else if $e_i < 0$ **then**
    $e_i(1) = -e_i, e_i(0) = 0$
    $k(0) \leftarrow k(0) \cdot \ell_i$
  else
    $e_i(0) = 0, e_i(1) = 0$
    $k(0) \leftarrow k(0) \cdot \ell_i$
    $k(1) \leftarrow k(1) \cdot \ell_i$
  end if
end for

Constant-time Group Action

Algorithm 3: Constant-time secret key decoding

```
for i = 0 to n − 1 do
    Set s ← 1 if e_i is negative, otherwise s ← 0.
    Set v ← 0 if e_i is 0, otherwise v ← 1.
    e_i(s) ← e_i − (2 · s · e_i).
    e_i(s̄) ← 0.
    k(s̄) ← ℓ_i · k(s̄).
    k(v̄) ← (ℓ_i − v · (ℓ_i − 1)) · k(v̄).
end for
```

- We adopted the same strategy to remove all the conditional statements using mask operations for the entire group action algorithm
- We removed all the *while* loops and replaced them with constant-time *for* loops with constant number of iterations.
- Further details on constant-time implementation can be found in our publicly available library.

Implementation Parameters

- All the finite field arithmetic are designed and developed using hand-written ARMv8 assembly.
- The proposed arithmetic library is also totally constant-time.
- Our library is publicly available at:
  `https://github.com/amirjalali65/armv8-csidh`
- The executables are benchmarked on real ARMv8-powered cellphones.
- Target devices:
  - Cortex-A57: Huawei Nexus 6P running Android 7.1.1
  - Cortex-A72: Google Pixel 2 running Android 8.1.0

Implementation Results

Table: Constant-time ladder

		Constant-time		Variable-time	
		Cortex-A57	Cortex-A72	Cortex-A57	Cortex-A72
Key validation	cc $\times 10^6$	-	-	38	23
	seconds	-	-	0.02	0.01
Group action	cc $\times 10^6$	30,459	28,872	624	552
	seconds	15.6	12.03	0.32	0.23
Total CSIDH	cc $\times 10^6$	61,054	57,912	1,326	1,224
	seconds	31.3	24.1	0.68	0.51

Table: Uniform but variant-time ladder

Operation	Cortex-A57	Cortex-A72
Group action	11,286 $\cdot 10^6$ cc 5.94 s	10,824 $\cdot 10^6$ cc 4.51 s

Conclusion and Future Work

- We proposed a constant-time implementation of CSIDH on ARMv8 processors.
- Our implementation is free of any *if* or *while* statement.
- We adopted a set of engineering techniques and heuristics to provide a fully constant-time and optimized implementation of CSIDH.
- The performance results using CT Montgomery ladder are very **slow**.
- Further optimization techniques are required to make CSIDH as a secure candidate for PQC.
- We plan to optimize our library further in the near future.

Thank You!