Efficient Post-Quantum Undeniable Signature on 64-bit ARM

Amir Jalali [1] Reza Azarderakhsh [1] Mehran Mozaffari-Kermani [2]

Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, FL, USA

Department of Computer Science and Engineering, University of South Florida, FL, USA

August 2017

Outline

Isogeny-based Crypto History

- The first suggestions to use isogenies in crypto by Couveignes in 1997

- Supersingular isogeny-based hash function by Charles, Lauter and Goren in 2005

- Isogeny-based public-key cryptosystems by Rostovtsev and Stolbunov in 2006

- The biggest impetus by David Jao and Luca De Feo in 2011.

Undeniable Signature and SIUS

- Undeniable Signature
  1. Invented by Chaum in 1989
  2. Allows the signer to choose to whom signatures are verified
  3. Interactive protocol between the signer and the verifier
  4. Applications: e-voting, e-auction, e-cash, ...

Undeniable Signature and SIUS

- Undeniable Signature
  1. Invented by Chaum in 1989
  2. Allows the signer to choose to whom signatures are verified
  3. Interactive protocol between the signer and the verifier
  4. Applications: e-voting, e-auction, e-cash, ...
- This work presents the first practical implementation of the Isogeny-based Undeniable Signature (SIUS) which was first introduced by Jao and Soukharev in 2014
  1. Smallest keys and signature size compared to other post-quantum candidates
  2. Fast and optimized implementation
  3. Quantum-resistant undeniable signature scheme

Isogenies on Elliptic Curves

Definition

Let E and E' be elliptic curves over $\mathbb{F}$.
An isogeny $\phi : E \rightarrow E'$ is a non-constant algebraic morphism (defined by polynomials)

$$\phi(x, y) = (\frac{p(x)}{q(x)}, \frac{s(x)}{t(x)} y)$$

satisfying $\phi(\infty) = \infty$ and $\phi(P + Q) = \phi(P) + \phi(Q)$.

The kernel H determines the image curve E' up to isomorphism

$$E/H := E'$$

$\deg(\phi)$ is its degree as an algebraic map

SIUS Overview

- Public Parameters
  - $p = \ell_A^{e_A}\ell_B^{e_B}\ell_C^{e_C}f \pm 1$, where ℓ_A, ℓ_B, and ℓ_C are small primes, e_A, e_B, and e_C 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 $\{P_A, Q_A\}$, $\{P_B, Q_B\}$, and $\{P_C, Q_C\}$ over $E_0[\ell_A^{e_A}]$, $E_0[\ell_B^{e_B}]$, and $E_0[\ell_C^{e_C}]$, respectively.

SIUS Overview

- Public Parameters
  - $p = \ell_A^{e_A}\ell_B^{e_B}\ell_C^{e_C} f \pm 1$, where ℓ_A, ℓ_B, and ℓ_C are small primes, e_A, e_B, and e_C 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 $\{P_A, Q_A\}$, $\{P_B, Q_B\}$, and $\{P_C, Q_C\}$ over $E_0[\ell_A^{e_A}]$, $E_0[\ell_B^{e_B}]$, and $E_0[\ell_C^{e_C}]$, respectively.
- Classical and quantum security is approximately $\sqrt[6]{p}$ and $\sqrt[9]{p}$, respectively.
  - Based on the difficulty of computing isogenies between supersingular elliptic curves.

SIUS Overview

- Key-generation:
  - The signer securely generates two random integers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ and computes $K_A = [m_A]P_A + [n_A]Q_A$
  - The signer computes isogeny map $\phi_A : E \to E_A/\langle K_A \rangle$ and also evaluates $\phi_A(P_C)$ and $\phi_A(Q_C)$ using ϕ_A.
  - The signer publishes the public-key as: E_A, $\phi_A(P_C)$, and $\phi_A(Q_C)$, while the private-key is (m_A, n_A).

SIUS Overview

- Key-generation:
  - The signer securely generates two random integers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ and computes $K_A = [m_A]P_A + [n_A]Q_A$
  - The signer computes isogeny map $\phi_A : E \to E_A/\langle K_A \rangle$ and also evaluates $\phi_A(P_C)$ and $\phi_A(Q_C)$ using ϕ_A.
  - The signer publishes the public-key as: E_A, $\phi_A(P_C)$, and $\phi_A(Q_C)$, while the private-key is (m_A, n_A).
- Signature:
  - The signer computes the message hash $h = H(M)$, $K_M = P_M + [h]Q_M$.
  - The signer first computes $\phi_M : E \to E_M = E/\langle K_M \rangle$

SIUS Overview

- Key-generation:
  - The signer securely generates two random integers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ and computes $K_A = [m_A]P_A + [n_A]Q_A$
  - The signer computes isogeny map $\phi_A : E \to E_A/\langle K_A \rangle$ and also evaluates $\phi_A(P_C)$ and $\phi_A(Q_C)$ using ϕ_A.
  - The signer publishes the public-key as: E_A, $\phi_A(P_C)$, and $\phi_A(Q_C)$, while the private-key is (m_A, n_A).
- Signature:
  - The signer computes the message hash $h = H(M)$, $K_M = P_M + [h]Q_M$.
  - The signer first computes $\phi_M : E \to E_M = E/\langle K_M \rangle$
- The signature:

$$[E_{AM}, \phi_{M,AM}(\phi_M(P_C)), \phi_{M,AM}(\phi_M(Q_C))]$$

Confirmation Protocol

- The signer secretly selects random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$ to blind the signature and computes $\phi_C, \phi_{C,MC}, \phi_{A,AC}, \phi_{MC,AMC}$

Confirmation Protocol

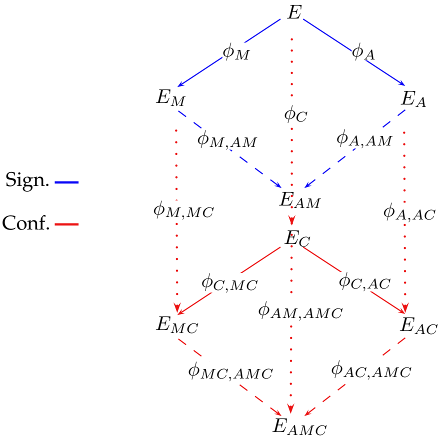
- The signer secretly selects random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$ to blind the signature and computes $\phi_C, \phi_{C,MC}, \phi_{A,AC}, \phi_{MC,AMC}$
- The signer commits E_C, E_{AC}, E_{MC}, E_{AMC}, and $\ker(\phi_{C,MC}) = \phi_C(K_M)$.
- The verifier randomly selects a bit $b \in \{0,1\}$

Confirmation Protocol

- The signer secretly selects random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$ to blind the signature and computes $\phi_C, \phi_{C,MC}, \phi_{A,AC}, \phi_{MC,AMC}$
- The signer commits E_C, E_{AC}, E_{MC}, E_{AMC}, and $\ker(\phi_{C,MC}) = \phi_C(K_M)$.
- The verifier randomly selects a bit $b \in \{0, 1\}$
- If $b = 0$
  - The signer outputs $\ker(\phi_C)$
  - The verifier computes $\ker(\phi_{A,AC})$, $\phi_{M,MC}$, $\phi_{AM,AMC}$, $\phi_{C,MC}$.
  - Verifier checks the correctness of all the committed information by signer.
- If $b = 1$
  - The signer outputs $\ker(\phi_{C,AC})$
  - The verifier computes $\phi_{MC,AMC}$,$\phi_{AC,AMC}$ and checks the corresponding curves in the commitment.

Confirmation Protocol

Figure: Signature and confirmation protocol in SIUS scheme



Disavowal Protocol

- The signer is presented with a fake signature (E_F, F_P, F_Q) instead of the real signature $(E_{AM}, \phi_{M,AM}(\phi_M(P_C)), \phi_{M,AM}(\phi_M(Q_C)))$

Disavowal Protocol

- The signer is presented with a fake signature (E_F, F_P, F_Q) instead of the real signature $(E_{AM}, \phi_{M,AM}(\phi_M(P_C)), \phi_{M,AM}(\phi_M(Q_C)))$
- The signer secretly selects random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$ along with all the curves and isogeny maps as shown before

Disavowal Protocol

- The signer is presented with a fake signature (E_F, F_P, F_Q) instead of the real signature $(E_{AM}, \phi_{M,AM}(\phi_M(P_C)), \phi_{M,AM}(\phi_M(Q_C)))$
- The signer secretly selects random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$ along with all the curves and isogeny maps as shown before
- The signer commits E_C, E_{AC}, E_{MC}, E_{AMC}, and $\ker(\phi_{C,MC}) = \phi_C(K_M)$

Disavowal Protocol

- The signer is presented with a fake signature (E_F, F_P, F_Q) instead of the real signature $(E_{AM}, \phi_{M,AM}(\phi_M(P_C)), \phi_{M,AM}(\phi_M(Q_C)))$
- The signer secretly selects random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$ along with all the curves and isogeny maps as shown before
- The signer commits E_C, E_{AC}, E_{MC}, E_{AMC}, and $\ker(\phi_{C,MC}) = \phi_C(K_M)$
- The verifier randomly generates a bit $b \in \{0, 1\}$

Disavowal Protocol

- The signer is presented with a fake signature (E_F, F_P, F_Q) instead of the real signature $(E_{AM}, \phi_{M,AM}(\phi_M(P_C)), \phi_{M,AM}(\phi_M(Q_C)))$
- The signer secretly selects random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$ along with all the curves and isogeny maps as shown before
- The signer commits E_C, E_{AC}, E_{MC}, E_{AMC}, and $\ker(\phi_{C,MC}) = \phi_C(K_M)$
- The verifier randomly generates a bit $b \in \{0, 1\}$
- The verifier computations are all the same as before except in case of $b = 0$ which requires one more isogeny computation: $\phi_F : E_F \rightarrow E_{FC} = E_F/\langle [m_C]F_P + [n_C]F_Q \rangle$.
- The verifier computes this isogeny and compares it with E_{AMC} (committed value by signer). These values should be different.

SIUS-Friendly Primes

- Smooth Isogeny Prime: $p = \ell_A^{e_A} \ell_B^{e_B} \ell_C^{e_C} \cdot f \pm 1$, where ℓ_A, ℓ_B, and ℓ_C are small primes, e_A, e_B, and e_C are positive integers, and f is a small cofactor to make the number prime

SIUS-Friendly Primes

- Smooth Isogeny Prime: $p = \ell_A^{e_A} \ell_B^{e_B} \ell_C^{e_C} \cdot f \pm 1$, where ℓ_A, ℓ_B, and ℓ_C are small primes, e_A, e_B, and e_C 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 $\ell_B = 3$ in affine and projective coordinates

SIUS-Friendly Primes

- Smooth Isogeny Prime: $p = \ell_A^{e_A} \ell_B^{e_B} \ell_C^{e_C} \cdot f \pm 1$, where ℓ_A, ℓ_B, and ℓ_C are small primes, e_A, e_B, and e_C 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 $\ell_B = 3$ in affine and projective coordinates
- We propose new set of formulas for $\ell_C = 5$ in projective coordinates

SIUS-Friendly Primes

- Smooth Isogeny Prime: $p = \ell_A^{e_A} \ell_B^{e_B} \ell_C^{e_C} \cdot f \pm 1$, where ℓ_A, ℓ_B, and ℓ_C are small primes, e_A, e_B, and e_C 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 $\ell_B = 3$ in affine and projective coordinates
- We propose new set of formulas for $\ell_C = 5$ in projective coordinates
- Security of a large-degree isogeny is $\sqrt[3]{\ell^e}$
  - Quantum claw finding problem by Childs in 2014.

SIUS-Friendly Primes

- Find two different primes at different security levels for a variety of optimizations

SIUS-Friendly Primes

- Find two different primes at different security levels for a variety of optimizations
- Prime search criteria:
  - Security:
    - The relative security of SIUS over a prime is based on $\min(\ell_A^a, \ell_B^b, \ell_C^c)$.

SIUS-Friendly Primes

- Find two different primes at different security levels for a variety of optimizations
- Prime search criteria:
  - Security:
    - The relative security of SIUS over a prime is based on $\min(\ell_A^a, \ell_B^b, \ell_C^c)$.
  - Speed:
    - Primes of the form $p = 2^a \ell_B^b \cdot f - 1 \rightarrow$ Montgomery-friendly property
    - Prime search: efficiency parameter θ for a prime of the form $p = \ell_A^{e_A} \ell_B^{e_B} \ell_C^{e_C} - 1$

$$\theta = \frac{\text{nbits}(p)}{\min(\text{nbits}(\ell_A^{e_A}, \ell_B^{e_B}, \ell_C^{e_C}))/3}$$

    - Recall: security of a large-degree isogeny is $\sqrt[3]{\ell^e}$
    - We are interested in the primes with the smaller value of θ

Proposed SIUS-Friendly Primes

Table: Proposed smooth implementation-friendly primes for SIUS scheme

$p = \ell_A^{e_A} \ell_B^{e_B} \ell_C^{e_C} - 1$	Prime size (bits)	Quantum Security	Classical Security	Prev. Signature (bytes)	Signature (bytes)
$2^{250}3^{163}5^{110} - 1$	764	83	125	764	573
$2^{330}3^{210}5^{151} - 1$	1014	110	165	1014	761

- By ignoring the curve coefficient B and using projective coordinates, each element of the signature, i.e., curve and auxiliary points is represented by only one field element in $\mathbb{F}_{p^2}$
- Therefore SIUS signature and public-key in our implementation are 25% smaller than the original signature sizes reported in the original scheme by Jao and Soukharev.

Projective Isogeny costs

- Projective 3 Isogenies
  1. Isogeny map: $(6M + 2S + 5a)$
  2. Isogeny eval.: $(3M + 3S + 8a)$
- Projective 4 Isogenies
  1. Isogeny map: $(5S + 7a)$
  2. Isogeny eval.: $(3M + 3S + 8a)$
- Projective 5 Isogenies
  1. Isogeny map: $(10M + 2S + 7a) \rightarrow$ slow
  2. Isogeny eval.: $(30M + 4S + 16a) \rightarrow$ very slow

Confirmation Protocol Mechanism

- Interactive procedure (both parties should involve)
- The verifier's computations depend on the value of b
- Disavowal protocol mechanism is almost the same

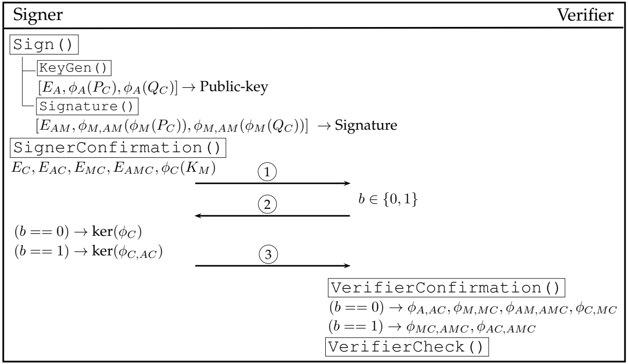


Figure: The SIUS confirmation protocol mechanism.

Finite-Field Arithmetic on ARMv8

- A64 or Advanced SIMD?

Finite-Field Arithmetic on ARMv8

- A64 or Advanced SIMD?
  - A64: General-purpose register file with thirty one 64-bit registers (radix-2^{64})

Finite-Field Arithmetic on ARMv8

- A64 or Advanced SIMD?
  - A64: General-purpose register file with thirty one 64-bit registers (radix-2^{64})
  - Adv. SIMD: 256-bit vectors which can be used to implement 32×32-bit multiplication in parallel (radix-2^{32})

Finite-Field Arithmetic on ARMv8

- A64 or Advanced SIMD?
  - A64: General-purpose register file with thirty one 64-bit registers (radix-2^{64})
  - Adv. SIMD: 256-bit vectors which can be used to implement 32×32-bit multiplication in parallel (radix-2^{32})
  - Both take the same number of multiplications for the implementation of field multi-precision multiplication
  - A64 implementation is faster because ASIMD multiplications are more expensive!

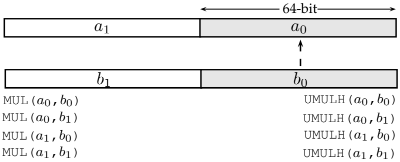


Figure: 8×A64 multiplications

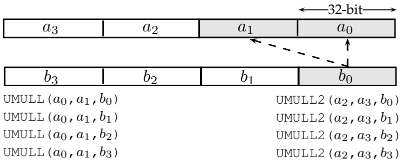


Figure: 8×ASIMD multiplications

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

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
  - Utilizes ARMv8 A64 registers in radix-2^{64} representation
  - 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
  - Also reduction over $\hat{p} = p + 1$ which eliminates several single-precision multiplications by "0" limbs:
    - $p764 + 1$ and $p1014 + 1$ have three and five 64-bit words equal to "0" in the lower half.

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 $O(\log^3 n)$

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 $O(\log^3 n)$
- Since we implemented the whole point arithmetic in projective coordinates, the number of filed inversions are scarce

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 $O(\log^3 n)$
- Since we implemented the whole point arithmetic in projective coordinates, the number of filed inversions are scarce
- We implemented constant-time FLT field inversion with fixed-window method
  - We prioritize security over a small amount of performance improvement in using non-constant time algorithms

Benchmark Targets

- The first empirical implementation of a quantum-resistant undeniable signature

Benchmark Targets

- The first empirical implementation of a quantum-resistant undeniable signature
- Target processor: Huawei Nexus 6P smart phone with a 2.0 GHz Cortex-A57 and a 1.55 GHz Cortex-A53 processors running Android 7.1.1

Benchmark Targets

- The first empirical implementation of a quantum-resistant undeniable signature
- Target processor: Huawei Nexus 6P smart phone with a 2.0 GHz Cortex-A57 and a 1.55 GHz Cortex-A53 processors running Android 7.1.1
- Portable version is benchmarked on:
  - 2.3 GHz NVIDIA Jetson TK1 equipped with a 32-bit ARMv7 Cortex-A15 running Ubuntu 14.04 LTS
  - 2.1 GHz Intel x64 i7-6700 running Ubuntu 16.04 LTS

Results

- Verifier's operations (server-side) are more computationally intensive
  - Performance bottleneck → $b = 0$
- More efficient degree 5 isogenies formulas → significant performance improvement (future work)

Table: Performance results ($\times 10^6$ CPU clock cycles)

Field Size	PQ Security	Lang.	Keygen Sign.	Signer Conf. / Disv.	Verifier ($b=0$) Conf.	Verifier ($b=0$) Disv.	Verifier ($b=1$) Conf. / Disv.
Huawei Nexus 6P ARMv8-A57 at 2.0 GHz							
764	83	C	1,068	1,416	2,638	2,980	1,138
		ASM	230	290	544	614	232
1014	110	C	2,646	3,592	6,854	7,726	2,918
		ASM	512	684	1,310	1,466	552
Huawei Nexus 6P ARMv8-A53 at 1.55 GHz							
764	83	C	2,024	2,595	4,834	5,463	2,085
		ASM	516	652	1,213	1,378	549
1014	110	C	4,515	6,142	11,724	13,153	4,972
		ASM	1,227	1,671	3,199	3,585	1,350
Desktop PC Intel x64 i7-6700 at 2.1 GHz							
764	83	C	493	655	1,222	1379	684
1014	110	C	1,136	1,545	2,973	3,357	1,623
NVIDIA Jetson TK1 ARMv7-A15 at 2.3 GHz							
764	83	C	3,433	4,549	8,473	9,574	3,657
1014	110	C	8,052	10,957	20,913	23,453	8,868

Conclusions

- **Efficient** implementation of SIUS on ARMv8 platforms
- Proposed SIUS-friendly primes with an efficiency parameter
- Hand-optimized finite-field arithmetic $\rightarrow$ up to 5 times faster than generic C implementation
- Analysis of the ARMv8 capabilities for finite field arithmetic implementation
- Implementations on Huawei Nexus 6P $\rightarrow$ practical benchmark on a smart phone
- We reduce the signature and public-key sizes of SIUS protocol by 25% compared to the original scheme
- Thank you!