Efficient Post-Quantum Undeniable Signature on 64-bit ARM

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

[1] Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, FL, USA,
{ajalali2016, razarderakhsh}@fau.edu
[2] Department of Computer Science and Engineering, University of South Florida, FL, USA,
mmozaff@gmail.com

Abstract. We present a full-fledged, highly-optimized, constant-time software for post-quantum supersingular isogeny-based undeniable signature (SIUS) on the ARMv8 platforms providing 83- and 110-bit quantum security levels. To the best of our knowledge, this work is the first empirical implementation of isogeny-based quantum-resistant undeniable signature presented to date. The proposed software is developed on the top of our optimized hand-written ARMv8 assembly arithmetic library and benchmarked on a variety of platforms. The entire protocol runs less than a second on Huawei Nexus smart phone, providing 83-bit quantum security level. Moreover, our signature and public key sizes are 25% smaller than the original SIUS scheme. We remark that the SIUS protocol, similar to other isogeny-based schemes, suffers from the excessive number of operations, affecting its overall performance. Nonetheless, its significantly smaller key and signature sizes make it a promising candidate for post-quantum cryptography.

Keywords: ARM assembly, supersingular isogeny-based cryptosystem, undeniable signature

1 Introduction

To prepare for the advent of quantum computers, the state-of-the-art research work has been investigating various public-key cryptography primitives which are assumed to be resistant against Shor's quantum algorithm [?]. One family of these primitives is based on the hardness of computing isogenies between two isogenous supersingular elliptic curves. Elliptic curve isogenies were first proposed by Couveignes [?] as an alternative underlying problem of elliptic curve cryptography. Construction of public-key cryptography from the isogeny of regular elliptic curves was introduced by Rostovtsev and Stolbunov [?,?]. However, the proposed scheme was later found to be unassured due to the sub-exponential quantum attack proposed by Childs et al. [?]. Cryptographic schemes based on supersingular elliptic curve isogenies were also applied in cryptographic hash

functions by Charles-Lauter-Goren [?] which proposed the hardness of path-finding in supersingular isogeny graphs. Isogenies on elliptic curves have been used as an assumption for other cryptographic systems such as Diffie-Hellman key-exchange [?], authenticated encryption [?], and signatures [?,?,?]. To date, the best known classical and quantum attacks against the supersingular isogeny problem have exponential complexity, making this cryptosystem to be one of the auspicious quantum-resistant candidates. Furthermore, isogeny-based schemes are constructed over elliptic curves and provide significantly smaller key size compared to other quantum-resistant candidates. This is desirable for the applications where communication bandwidth is restricted. Recently, it is pointed out that isogeny-based cryptosystems can be utilized with even smaller keys using key compression techniques [?,?].

Recent attempts to efficiently implement isogeny-based key-exchange protocol, in software [?,?,?] and hardware [?], show that this cryptography primitive can be efficiently implemented on different platforms with reasonable performance metrics. However, the performance evaluation of other supersingular isogeny-based schemes such as undeniable signature has not been investigated in depth. In this work, we present a constant-time software for the signature and confirmation/disavowal operations of supersingular isogeny-based undeniable signature (SIUS) which was first introduced by Jao and Soukharev [?]. Furthermore, we benchmark our software on a variety of platforms to evaluate the performance of a quantum-resistant undeniable signature as a reference. Additionally, we develop an optimized version of the SIUS scheme for the 64-bit ARM platforms with a special focus on the ARMv8 Cortex-A57 processor. The proposed implementation is developed based on the projective coordinates and curve coefficients in analogy with the projective formulas which are proposed in [?]. We plan to make our software publicly available in the near future.

The main contributions of this paper are summarized as follows:

- We propose a new set of *inversion-free* projective formulas for computing degree 5 isogenies of supersingular Montgomery curves. Previous implementations of isogeny-based cryptosystems mainly focused on Diffie-Hellman key exchange protocol (SIDH) which is constructed over the two subgroups of points on elliptic curves; accordingly, efficient formulas for 3 and 4 degree isogenies have been studied and implemented in [?,?,?]. However, since the isogeny-based undeniable signature is constructed on three such subgroups of points, in this work, we develop projective degree 5 isogenies formulae and implement them efficiently on our target processor.
- Taking advantage of reduced curve coefficient technique in Kummer varieties, we reduce the signature and public-key sizes of SIUS protocol by 25% compared to the original definition of this protocol in [?].
- We introduce two *implementation-friendly* primes for different quantum security levels. The proposed primes have a special shape that can be used to efficiently implement isogenies and finite field arithmetic computations on 64-bit platforms. We include a comparative discussion of implementa-

tion techniques on the ARMv8-A platforms based on their capabilities to efficiently implement finite field arithmetic.
- We implement the SIUS protocol in C language for two quantum-security levels. The presented implementation is portable on different platforms, providing 83 and 110 bits of quantum security. We also present an optimized version of the protocol for the ARMv8-A platforms. To the best of our knowledge, our software is the first implementation of the SIUS found in the literature.

2 Preliminaries

This section provides a brief overview of the isogeny-based undeniable signature scheme and its features. We refer readers to [?,?,?] for more detailed information of quantum-resistant isogeny-based cryptography and its related protocols.

2.1 Isogenies and Kernels

Let E_1 and E_2 be elliptic curves over a field $\mathbb{K}$. An isogeny over $\mathbb{K}$ is a rational map over $\mathbb{K}$ which is denoted as $\phi : E_1 \to E_2$ such that $\phi(\mathcal{O}_{E_1}) = \mathcal{O}_{E_2}$. The degree of an isogeny, denoted as ℓ, is the degree of its rational map. We represent the isogeny of degree ℓ as ℓ-isogeny. If there exists an isogeny of degree ℓ between two elliptic curves E_1 and E_2, then these two curves are ℓ-isogenous, and they share the same j-invariant value. Isogenies of elliptic curves are identified with their kernels using Vélu's formula [?]. The kernel of an isogeny ϕ of degree ℓ is a finite subgroup of points in $E(\overline{\mathbb{K}})$ and defined as: $\ker(\phi) = \{\mathcal{O}_E\} \cup \{P = (x_p, y_p) \in E(\overline{\mathbb{K}}) : \text{order}(P) = \ell\}$, and for a separable isogeny of degree ℓ has exactly ℓ elements. Let E be an elliptic curve defined over $\mathbb{K}$ and G a finite subgroup of $E(\overline{\mathbb{K}})$ which is defined over $\mathbb{K}$. Then, there is an isogenous elliptic curve $E' : E/\langle G \rangle$ and an isogeny map $\phi : E \to E'$ both defined over $\mathbb{K}$ with $\ker(\phi) = G$ [?]. In this work, all the kernels are cyclic groups and we can evaluate isogenies using the kernel or any single generator of the kernel. For small values of ℓ, we can compute this isogeny efficiently using Vélu's formula. Moreover, as it is discussed in details in [?,?,?,?], large-degree isogenies of smooth order elliptic curves can be computed using consecutive elliptic curve point multiplication and the evaluation of small-degree isogenies. The computation procedure adopts an optimal strategy which computes the leaves of the isogeny graph efficiently using a combination of point multiplication, isogeny evaluation, and divide-and-conquer method. However, the optimal strategy over a defined finite field depends on the cost of point multiplication by ℓ and ℓ-isogeny evaluation of elliptic curves on the target platform. We return to this discussion in Section ??.

2.2 Supersingular Isogeny Undeniable Signature

The undeniable signature was first introduced by Chaum et al. [?] which was constructed based on discrete logarithm problem. Furthermore, the security of this scheme was defined by Kurosawa et al. [?], in which the invisibility concept

of undeniable signatures was characterized. Unlike a digital signature, an undeniable signature requires an interactive procedure between signer and verifier to confirm and disavow valid and forged signatures, respectively. It is noted that any undeniable signature scheme requires 6 specific functions to securely generate, verify, and disavow a signature. These functions have been first defined in [?] and denoted as:

$$\sum = (G_{\mathrm{k}}, S, V, S_{\mathrm{sim}}, \pi_{\mathrm{con}}, \pi_{\mathrm{dis}}),$$

where a key generation algorithm G_{k}, a signature algorithm S, a validity check V, a signature simulator S_{sim}, a confirmation protocol π_{con}, and finally a disavowal protocol π_{dis} make up an undeniable signature scheme. The confirmation protocol π_{con} and the disavowal protocol π_{dis} are used by signer to prove to the verifier that the signature is valid or invalid, respectively. Moreover, an undeniable signature scheme is assumed to be secure, if and only if it completely satisfies unforgeability and invisibility [?]. We refer to [?,?] for details on the definitions of unforgeability and invisibility.

SIUS is defined over smooth primes of the form $p = \ell_A^{e_A}\ell_B^{e_B}\ell_C^{e_C}.f \pm 1$, where ℓ_A, ℓ_B, and ℓ_C are small primes and f is a small factor. A supersingular elliptic curve E of cardinality $\#E = (p \mp 1)^2 = (\ell_A^{e_A}\ell_B^{e_B}\ell_C^{e_C}.f)^2$ can be constructed over $\mathbb{F}_{p^2}$ using Bröker's algorithm [?] which is the SIUS scheme base curve, and its coefficients are public parameters. Furthermore, three pairs of random points on E denoted as $\{P_A, Q_A\} \in E[\ell_A^{e_A}]$, $\{P_M, Q_M\} \in E[\ell_B^{e_B}]$, and $\{P_C, Q_C\} \in E[\ell_C^{e_C}]$ are randomly chosen as the starting points. Hence, the protocol public parameters are p, E, $\{P_A, Q_A\}$, $\{P_M, Q_M\}$, $\{P_C, Q_C\}$, and a hash function H which is used to compute the message hash before the signing procedure.

Signature. The signer securely generates two random integers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$, computes the point $K_A = [m_A]P_A + [n_A]Q_A$ on elliptic curve E, and gets the isogenous curve E_A using $\ell_A^{e_A}$-isogeny map $\phi_A : E \to E_A/\langle K_A\rangle$. The signer also evaluates $\phi_A(P_C)$ and $\phi_A(Q_C)$ using ϕ_A and 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). The signer computes the message hash $h = H(M)$, $K_M = P_M + [h]Q_M$, and sets it as the kernel of isogeny ϕ_M. Moreover, the signer computes $\phi_M(K_A)$ and $\phi_A(K_M)$ which are the kernel of the isogeny $\phi_{M,AM}$ and $\phi_{A,AM}$, respectively. In order to generate the signature, the signer computes the following isogenies:

- $\phi_M : E \to E_M = E/\langle K_M\rangle$,
- $\phi_{M,AM} : E_M \to E_{AM} = E_M/\langle\phi_M(K_A)\rangle \cong E_A/\langle\phi_A(K_M)\rangle$.

Figure ?? illustrates the corresponding required maps to generate the signature E_{AM} from the base curve E. Additionally, using $\phi_{M,AM}$, the signer evaluates $\phi_{M,AM}(\phi_M(P_C))$ and $\phi_{M,AM}(\phi_M(Q_C))$ on E_{AM}, and presents these two points along with E_{AM} as the signature string.

Confirmation Protocol π_{con}. To confirm the signature, E_{AM} should be confirmed without disclosing the signature isogenies, i.e., $\phi_{M,AM}$ and $\phi_{A,AM}$. To

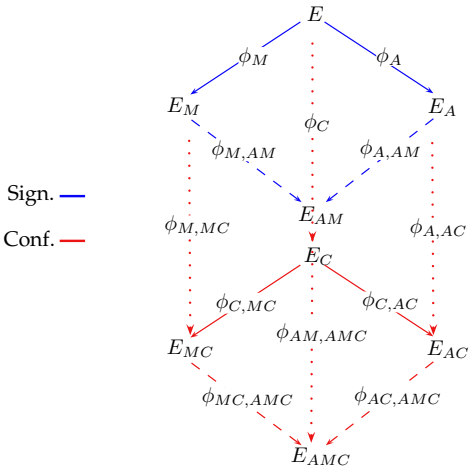

Fig. 1: Signature (Sign.) and Confirmation (Conf.) protocol isogeny maps.

this end, signer uses the public points $\{P_C, Q_C\}$ and generates another isogeny ϕ_C similar to ϕ_A:

1. The signer generates two secret integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ and computes the kernel $K_C = [m_C]P_C + [n_C]Q_C$. Consecutively, the signer computes the following isogenies:

- $\phi_C : E \to E_C = E/\langle K_C \rangle$,
- $\phi_{C,MC} : E_C \to E_{MC} = E_C/\langle \phi_C(K_M) \rangle$,
- $\phi_{A,AC} : E_C \to E_{AC} = E_A/\langle \phi_A(K_C) \rangle$,
- $\phi_{MC,AMC} : E_{MC} \to E_{AMC} = E_{MC}/\langle \phi_{C,MC}(K_A) \rangle$.

The signer further commits E_C, E_{AC}, E_{MC}, E_{AMC}, and $\ker(\phi_{C,MC}) = \phi_C(K_M)$ to be verified. Note that here, the signer uses $\{P_C, Q_C\}$ to eventually blind the signature E_{AM} through E_{AMC} as a commitment without disclosing the required information to compute the actual signature.

2. The verifier randomly generates a bit $b \in \{0, 1\}$ and sends it to the signer:
   (a) If $b = 0$, the signer outputs $\ker(\phi_C) = K_C$. Since E_A is available in the signer's public-key, the verifier is able to compute $\ker(\phi_{A,AC})$. Moreover, using $\ker(\phi_M) = K_M$, the verifier can compute $\ker(\phi_{M,MC}) = \phi_M(K_C)$. The verifier uses the auxiliary points in the signature, i.e., $\phi_{M,AM}(\phi_M(P_C)$ and $\phi_{M,AM}(\phi_M(Q_C))$, and computes $\phi_{AM,AMC}$. Finally, verifier utilizes the signer's output point $\ker(\phi_C)$ and K_M, and verifies $\ker(\phi_{C,MC}) = \phi_C(K_M)$ which is committed by the signer. The verifier checks that all the computed kernels map between the corresponding curves specified in the signer's commitment. Note that the verification procedure is performed simply by comparing the j-invariant values of the curves.

(b) If $b = 1$, the signer outputs $\ker(\phi_{C,AC}) = \phi_C(K_A)$. Using this value, the verifier computes $\phi_{MC,AMC}$ and $\phi_{AC,AMC}$, and verifies if $\phi_{C,AC}$, $\phi_{MC,AMC}$, and $\phi_{AC,AMC}$ correctly map between the coresponding committed curves by the signer.

Disavowal Protocol $\pi_{\mathbf{dis}}$. In disavowal protocol, given a falsified signature, the signer wishes to convince the verifier that the presented signature is fake. In this case, 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 should disavow E_F without revealing any credentials such as E_{AM}. To this end, the signer, similar to confirmation protocol, exploits the point $\{P_C, Q_C\}$ to blind E_{AM}, yet gives the verifier enough information that the verifier can compute E_{FC} and check that $E_{FC} \neq E_{AMC}$.

1. The signer generates two secret random integers $m_C, n_C \in \mathbb{Z}/\ell_C^{e_C}\mathbb{Z}$ to compute $\ker(\phi_C) = K_C = [m_C]P_C + [n_C]Q_C$. The signer computes all the required kernels and isogenies to blind E_{AM} using E_{AMC} similar to Step 1 in the confirmation protocol π_{con}. The signer commits E_C, E_{AC}, E_{MC}, E_{AMC}, and $\ker(\phi_{C,MC}) = \phi_C(K_M)$.
2. The verifier selects $b \in \{0, 1\}$:
   (a) If $b = 0$, the signer provides $\ker(\phi_C)$. The verifier computes $\ker(\phi_C)$, $\ker(\phi_{M,MC})$, and $\ker(\phi_{A,AC})$ using $\ker(\phi_C)$. Also, the verifier computes $\ker(\phi_{C,MC})$ independently and checks its value with the commitment. Using knowledge of E_F (fake signature), the verifier computes the isogeny map $\phi_{F,FC} : E_F \to E_{FC} = E_F/\langle [m_C]F_P + [n_C]F_Q \rangle$. Now, the verifier has all the required isogeny maps to check the correctness of the corresponding curves in the signer's commitment as well as checking that $E_{FC} \neq E_{AMC}$.
   (b) If $b = 1$, the signer outputs $\ker(\phi_{C,AC})$. The verifier computes $\phi_{MC,AMC}$ and $\phi_{AC,AMC}$, and checks if $\phi_{C,AC}$, $\phi_{MC,AMC}$, and $\phi_{AC,AMC}$ map the corresponding committed curves correctly similar to confirmation protocol.

3 Implementation Parameters

Unlike traditional elliptic curve cryptography with a fixed curve, isogeny-based cryptosystem computes the isogeny between different curves and maps the corresponding points which are computationally intensive for large-degree isogenies. Hence, from the first version of isogeny-based software (Diffie-Hellman key exchange scheme) developed by De Feo et al. [**?**], all the required arithmetic of elliptic curves were computed in Kummer varieties using Montgomery arithmetic, taking advantage of their efficient computations. Moreover, the recently proposed projective formulas for isogeny computations [**?**] set the performance bar higher and provide faster, yet constant-time library for SIDH key exchange scheme by providing almost inversion-free implementation. In this work, we follow the same methodology and arithmetic for the isogeny computations to achieve efficient performance results.

3.1 Projective Isogenies of Montgomery Curves

We follow the implementation parameters and strategies described in [?] for 3- and 4-isogeny computations, while we propose new sets of projective formulas for 5-isogeny computations on Montgomery curves.

Let $E : by^2 = x^3 + ax^2 + x$ be a Montgomery curve defined over a field $\mathbb{K}$ not of characteristic 2, where $a, b \in \mathbb{K}$ and $a(b^2 - 4) \neq 0$. The projective points on E are all points $(X : Y : Z) \in \mathbb{P}^2(\mathbb{K}) = \{(X : Y : Z) : (X, Y, Z) \in \mathbb{K}^3 - \{(0, 0, 0)\}\}$ satisfying the homogeneous equation:

$$bZY^2 = X^3 + aZX^2 + Z^2X.$$

Moreover, we can convert the curve coefficients to projective coordinates as $(A : B : C) \in \mathbb{P}^2(K)$, where $a = A/C$ and $b = B/C$. Now, the fully projective curve equation is:

$$BZY^2 = CX^3 + AZX^2 + Z^2CX.$$

Moreover, based on [?], isogeny and point arithmetic computations can be stated even more simply by ignoring B, since Kummer arithmetic is independent of this coefficient [?], and works solely with $(A : C) \in \mathbb{P}^1$. Based on these assumptions, we restate the Montgomery curves projective 3- and 4-isogeny formulae from [?] and [?], and develop new sets of formulas for projective 5 isogenies in the following.

Projective 3 Isogenies. An isogeny of degree ℓ can be efficiently computed for small values of ℓ using Vélu's formula and its kernel. For 3 isogenies, the kernel of the isogeny is the subgroup of points on E which has order 3. We denote this subgroup as $G_3 = \{P_3, -P_3, \mathcal{O}\}$ where $P_3 = (X_3 : Z_3) \in \mathbb{P}^1$ is a point with order equal to 3 on E. In analogy with the computations in [?] and [?], the projective 3-isogeny map $\phi_3 : E_{(A:C)} \to E'_{(A':C')}$, and 3-isogeny evaluation formulas $(X : Z) \mapsto (X' : Z')$ can be efficiently computed as

$$\phi_3 : (A' : C') = (Z_3^4 + 18X_3^2Z_3^2 - 27X_3^4 : 4X_3Z_3^3),$$

$$(X' : Z') = (X(X_3X - Z_3Z)^2 : Z(Z_3X - X_3Z)^2),$$

which cost $6\mathbf{M} + 2\mathbf{S} + 5\mathbf{a}$ for each isogeny map and $3\mathbf{M} + 3\mathbf{S} + 8\mathbf{a}$ for each evaluation.

Projective 4 Isogenies. Isogenies of degree four are constructed on the subgroup of the points on E which have the exact order equal to four. Again, we use Vélu's formula to derive the rational maps and refer to [?] for projectivizing the isogeny map and evaluation formulas. The 4-isogeny map and evaluation set of formulas can be expressed as follows:

$$\phi_4 : (A' : C') = (2(2X_4^4 - Z_4^4) : Z_4^4),$$

$$(X' : Z') = (X(2X_4Z_4Z - X(X_4^2 + Z_4^2))(X_4X - Z_4Z)^2) : \\ Z(2X_4Z_4X - Z(X_4^2 + Z_4^2))(Z_4X - X_4Z)^2),$$

where $P_4 = (X_4 : Z_4) \in \mathbb{P}^1$ is a 4-torsion point on E. The above formulas can be computed using $5\mathbf{S} + 7\mathbf{a}$ for isogeny map, and $9\mathbf{M} + 1\mathbf{S} + 6\mathbf{a}$ for isogeny evaluation using pre-computed coefficients $X_4^2 + Z_4^2$, $X_4^2 - Z_4^2$, $2X_4Z_4$, X_4^4, and Z_4^4 which are stored when the isogeny map ϕ_4 is computed.

Projective 5 Isogenies. Isogenies of degree 5, unlike the isogenies of degree 4 and degree 3, require more complicated set of formulas. First, we should construct the kernel using the subgroup of order 5 on E. Suppose $P_5 = (X_5 : Z_5) \in \mathbb{P}^1$ is a 5-torsion point on E and let $2P_5 = (\bar{X}_5 : \bar{Z}_5) \in \mathbb{P}^1$. The 5-torsion subgroup for computing isogeny can be represented as $G_5 = \{-2P_5, -P_5, \mathcal{O}, P_5, 2P_5\}$ which has exactly 5 elements. Applying the abscissas of P_5 and $2P_5$, we develop a set of formulas for computing 5-isogeny map and evaluating this isogeny for a given point $(X : Z)$.

For the 5-isogeny map, we use the fact that the x abscissas of P_5 and $[4]P_5 = [2]2P_5$ are equal. Using 5-division polynomials $\psi_5(x)$, the 5-isogeny map can be computed as:

$$\phi_5 : (A' : C') = (\bar{X}_5^4 Z_5 - 4X_5\bar{X}_5\bar{Z}_5(\bar{X}_5^2 + \bar{Z}_5^2) - Z_5\bar{Z}_5^2(2\bar{X}_5^2 - \bar{Z}_5^2) : 4X_5\bar{X}_5^2\bar{Z}_5^2)$$

using $10\mathbf{M} + 2\mathbf{S} + 7\mathbf{a}$, when the abscissa of $2P_5$ is available. For the isogeny evaluation, computations are more complex. Particularly, we notice that the Vélu's formula for computation of the 5-isogeny map leads to an unwieldy formula compared to 3 and 4 isogenies. The projective version of the 5-isogeny evaluation can be computed using

$$\begin{aligned}(X' : Z') = (&XZ_5\bar{Z}_5(X_5Z - XZ_5)^2(\bar{X}_5Z - X\bar{Z}_5)^2 + \\ &2Z[2Z^2(X_5\bar{Z}_5(\bar{X}_5Z - X\bar{Z}_5)^2(AX_5Z_5 + C(X_5^2 + Z_5^2)) + \\ &\bar{X}_5Z_5(X_5Z - XZ_5)^2(A\bar{X}_5\bar{Z}_5 + C(\bar{X}_5^2 + \bar{Z}_5^2))) + \\ &\bar{Z}_5(X_5Z - XZ_5)(\bar{X}_5Z - X\bar{Z}_5)^2(2AX_5Z_5 + C(3X_5^2 + Z_5^2)) + \\ &Z_5(\bar{X}_5Z - X\bar{Z}_5)(X_5Z - XZ_5)^2(2A\bar{X}_5\bar{Z}_5 + C(3\bar{X}_5^2 + \bar{Z}_5^2))] : \\ &CZZ_5\bar{Z}_5(X_5Z - XZ_5)^2(\bar{X}_5Z - X\bar{Z}_5)^2),\end{aligned}$$

which is more complicated than the 5-isogeny map; however, in our implementation, we store five coefficients during the computation of 5-isogeny map which are used in 5-isogeny evaluation. These coefficients are $\bar{X}_5Z_5$, $\bar{X}_5\bar{Z}_5$, $(\bar{X}_5^2 + \bar{Z}_5^2)$, $Z_5\bar{Z}_5$, and $\bar{X}_5^2$. Using these pre-computed values, the 5-isogeny can be evaluated in $30\mathbf{M} + 4\mathbf{S} + 16\mathbf{a}$. We state that the 5-isogeny evaluation formula in affine coordinates has relatively simpler formula than projective form; however, affine formulas require excessive number of field inversions which result in significant overall performance degradation if the inversions are computed using constant-time algorithms. Alternatively, non-constant time inversion algorithms can be

Table 1: 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)	$\min(\ell_A^{e_A}, \ell_B^{e_B}, \ell_C^{e_C})$ (bits)	θ	Quantum Security	Classical Security	Signature (bytes)
$2^{250}3^{163}5^{110} - 1$	764	251	9.13	83	125	573
$2^{330}3^{210}5^{151} - 1$	1014	331	9.19	110	165	761

deployed to implement the whole protocol in affine coordinates. Nevertheless, in such case, the software would be vulnerable to timing analysis attacks. Hence, we choose to work with projective coordinates, providing a constant-time software which is assumed to be secure against these types of attack.

3.2 Proposed Implementation-Friendly Primes

The SIUS scheme is built over a prime of the smooth form $p = \ell_A^{e_A}\ell_B^{e_B}\ell_C^{e_C}.f \pm 1$, taking advantage of its special shape to construct three different subgroups of points on E, i.e., $E[\ell_A^{e_A}]$, $E[\ell_B^{e_B}]$, and $E[\ell_C^{e_C}]$. Finding the efficient primes of this form is directly related to the field arithmetic algorithms and implementation platform architecture. Since we utilize Montgomery arithmetic, we choose to set $\ell_A = 2$ to find *Montgomery-friendly* primes ($p' = -p^{-1} \bmod R = 1$) [**?**]. The generic Montgomery reduction requires $s^2 + s$ multiplications, while reduction over Montgomery-friendly primes can be efficiently computed using s^2 multiplications for a $2s$-limb element.

Moreover, as it is discussed in [**?**], Montgomery reduction can be implemented even more efficiently for the primes of the form $p = 2^{e_A}\alpha - 1$, since it can be implemented based on multiplication of the finite field elements with $\hat{p} = p+1 = 2^{e_A}\alpha$ which has exactly $\lfloor \frac{e_A}{r} \rfloor$ least significant words equal to "0" in 2^r-radix representation; therefore, multiplication of these limbs can simply be neglected inside the reduction implementation. This implies that the larger values of e_A lead to even more efficient implementation of Montgomery reduction for the primes of this form, because the number of "0" words are increased. We return to this discussion in Section **??**.

So far, we set $\ell_A = 2$ and seek for the large values of e_A to make the reduction procedure more optimized. We also choose $f = 1$ since the SIUS security level depends only on the size of the kernels, and larger values of f do not provide any more security, yet increase the prime size. Furthermore, we set $\ell_B = 3$ and $\ell_C = 5$ to compute small-degree isogenies of elliptic curves efficiently using Velús formula. Moreover, as stated in [**?**], the fastest known quantum algorithm against the SIUS scheme require $O(n^{1/3})$ running time, where n is the size of the kernel; therefore, we search for the primes which provide reasonable level of quantum security, but not too large in size, so we can implement the finite field arithmetic efficiently on the ARM-powered devices. We propose an efficiency parameter to ease the prime search procedure of the SIUS smooth primes. Let

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

Table 2: Comparative timings for multiplication and isogeny evaluation in projective Kummer coordinates in terms of microseconds on ARMv8 Cortex-A57

Operation	$p764$			$p1014$		
	$\ell=3$	$\ell=4$	$\ell=5$	$\ell=3$	$\ell=4$	$\ell=5$
multiplication by ℓ (μs)	56	52	68	94	87	115
ℓ-isogeny evaluation (μs)	35	47	185	59	78	309
$r=$ mul/eval	1.6	1.1	0.3	1.6	1.1	0.3

be the efficiency parameter for a prime of the form $\ell_A^{e_A}\ell_B^{e_B}\ell_C^{e_C}-1$, where $\text{nbits}(n) = \lceil \log_2^n \rceil$ which represents the number of bits in n. In particular, we are interested in the primes with the smaller value of θ, so we attain higher level of security with smaller number of bits. For all the smooth primes of the form $p = \ell_A^{e_A}\ell_B^{e_B}\ell_C^{e_C} - 1$ with different size and security levels, this parameter is bounded by $9 < \theta < 10$ which makes it a reasonable measurement with low variation for the prime search procedure. We also choose the primes with the number of bits smaller than multiple of 64-bit word, so we can adopt a combination of Karatsuba multiplication, carry-handling elimination, and lazy reduction in $\mathbb{F}_{p^2}$ arithmetic for achieving better performance.

Based on the above assumptions, we search for the implementation-friendly primes which are well-fitted into our library and target processor. Table **??** includes our proposed primes for two different quantum security levels. We also ensure that these primes satisfy the security balance for computing isogenies of torsion subgroups, i.e., $\ell_A^{e_A}$, $\ell_B^{e_B}$, and $\ell_C^{e_C}$ have less than 40 bits difference pairwise.

Smaller Signature. We denote that 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 $\mathbb{F}_{p^2}$ which makes the SIUS signature and public-key in our implementation about 25% smaller than the original signature sizes reported in [**?**] for different security levels. This concept was first used in [**?**], providing smaller public-keys for the SIDH protocol.

3.3 Optimal Strategy for Large-degree Isogeny Computation

In the previous sections, we have described all the necessary formulas for computing small-degree isogenies. However, eventually, we require to compute smooth large-degree ℓ^e isogenies inside the protocol. This can be done by the composition of small-degree ℓ isogeny e times as $\phi = \phi_{e-1} \circ \phi_{e-2} \circ \cdots \circ \phi_0$ using different strategies. As it is pointed out in [**?**], we can demonstrate the computational structure of isogeny map between different points of elliptic curves as a graph, where left edges represent point multiplications by ℓ and right edges are ℓ-isogeny evaluations. Additionally, since multiplications and isogeny computations have different costs, different weights are assigned to the left and right edges of the graph. Jao et al. [**?**] developed an optimal strategy for the large-degree isogeny

computations by traversing this weighted graph at each point based on a decision algorithm. Their proposed strategy reveals the most efficient steps of computing large smooth degree isogenies. We adopt the same strategy in our implementation. Note that the cost of point multiplication by ℓ and ℓ-isogeny evaluation is different for each degree, i.e., ℓ_A, ℓ_B, and ℓ_C, as well for the target platform. We obtain these weights for each small-degree $\ell_A = 3$, $\ell_B = 4$, and $\ell_C = 5$ on our target processor and find the optimal strategy based on them. Table **??** includes the operation costs of multiplication by ℓ and ℓ-isogeny evaluation for different ℓ over the two finite fields on an ARMv8 Cortex-A57 core.

The provided numbers are averaged over 10^4 iterations of the functions, and they are implemented based on our optimized assembly library. The r ratio represents the relative cost of point multiplication to isogeny evaluation for each degree ℓ. Regarding this ratio, the optimal strategy traversal for each degree is computed. We observe that the smaller value of r leads to less number of isogeny evaluation operations in the final strategy.

3.4 Protocol Implementation

We implement the SIUS protocol using five main procedures:

1. `Sign()`: Key generation and signature operations performed by the signer.
2. `SignerConfirmation()`: The isogeny computations performed by the signer to commit the required curves and points.
3. `VerifierConfirmation()`: The isogeny computations performed by the verifier to confirm the correctness of a signature.
4. `SignerDisavowal()`: The isogeny computations performed by the signer to disavow a forged signature. These computations are identical to the signer's confirmation protocol.
5. `VerifierDisavowal()`: The isogeny computations performed by the verifier to check that the fake signature is disavowed by the signer.

Moreover, we implement the verifier's confirmation and disavowal functions based on the input bit $b \in \{0, 1\}$. Therefore, the number of operations and isogeny computations in verifier's confirmation and disavowal protocols depends on the b value. In Section **??**, we provide the corresponding timings for each function based on this value in detail. We also remark that in our implementation, all the verification operations are implemented by checking the j-invariant values of committed curves and the curves which are computed by the verifier using the public parameters.

Figure **??** illustrates the SIUS confirmation protocol mechanism based on the above functions. The same mechanism applies to the disavowal protocol using `SignerDisavowal()` and `VerfierDisavowal()` functions. Note that the verifier's disavowal protocol in case of $b = 0$ requires one more isogeny computation, i.e., $\phi_F : E_F \rightarrow E_{FC} = E_F/\langle [m_C]F_P + [n_C]F_Q \rangle$.

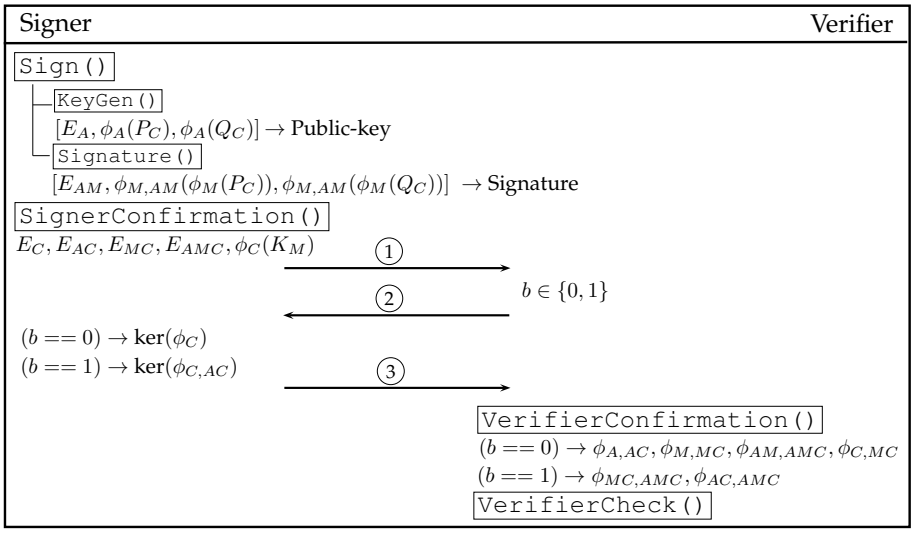

Fig. 2: The SIUS confirmation protocol mechanism.

4 $\mathbb{F}_p$ Arithmetic on ARMv8

We implement optimized field arithmetic library, targeting the ARMv8-A platform using AArch64 assembly instruction set. We concentrate on the development of tailored hand-optimized arithmetic functions for each proposed finite field. We employ loop unrolling, full register allocation, and multiple load/store techniques to highly optimize our field arithmetic library.

4.1 Target Platform Architecture

The proposed software is benchmarked on various platforms; however, we optimized our finite filed arithmetic library for the 64-bit ARM-powered devices. We run our software on a Huawei Nexus 6P smart phone which is equipped with ARMv8 Cortex-A57 and ARMv8 Cortex-A53, providing ARM big.LITTLE technology[3]. ARMv8 processors are capable of performing arithmetic instructions using the A64 instruction set with general registers, as well as Advanced SIMD instructions with vectors. The instruction processing pipeline is composed of two phases. First, instructions are fetched and decoded in order into internal micro-operations. Then, micro-operations stall for their operands and assign execution to one of the execution pipelines [?]. We note that the high-performance Cortex-A57 cores have separate execution pipes for ASIMD and A64 operations in fully *out-of-order* phase which results in fast computational power, while Cortex-A53 cores make use of highly power-efficient 8-stage *in-order* pipeline. We analyze

[3] ARM big.LITTLE technology is a power optimization technology where high-performance cores are combined with power-efficient cores to provide power-performance efficient benchmarks.

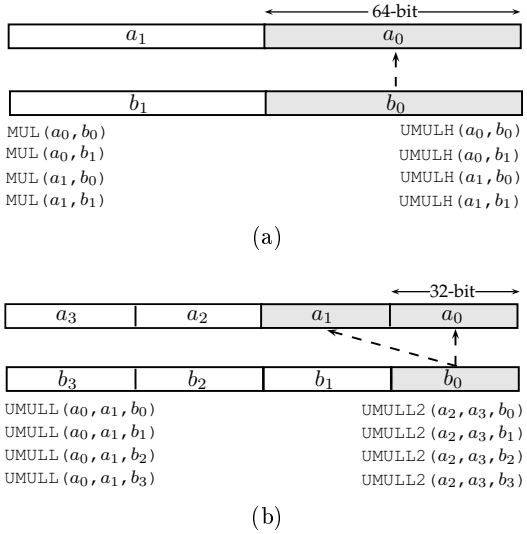

Fig. 3: (a) A64 and (b) Adv. SIMD 128×128-bit multiplication.

the usage of both instruction sets as well as their capabilities for field arithmetic implementation in the following:

A64 overview. The A64 instruction set provides similar functionality to the A32 and T32 instruction set in AArch32 for the ARMv7 platforms. However, it supports larger general-purpose register file with thirty one 64-bit unbanked registers [**?**]. This excessive number of registers is suitable for implementing field arithmetic over large fields, since the number of load and store instructions is infrequent. Moreover, the field operands can be represented in radix-2^{64} which translates into a significant improvement in performance compared to the previous family of 32-bit ARM processors. The A64 multiplication instructions, i.e., `MUL` and `UMULH`, take 4 and 6 clock cycles on Cortex-A57 processors, respectively [**?**]; the first instruction computes the low half of 64×64-bit multiplication result, while the latter one generates the high part.

Advanced SIMD. The AArch64 vector multiplication instruction is similar to ARMv7 NEON multiplication which computes two parallel 32×32-bit multiplication and generates a pair of 64-bit results. This operation takes roughly 6 clock cycles for the low half of the vector, i.e., `UMULL,` and 5 clock cycles for the upper half, i.e., `UMULL2,` on the Cortex-A57 cores, when there are no dependencies [**?**]. Moreover, since data are decomposed into 32-bit limbs, the implementation of arithmetic using Adv. SIMD instructions set leads to the representation of data in radix-2^{32}. This simply implies that the number of multiplication instructions is twice compared to radix-2^{64} representation. However, since a pair of 32×32-bit multiplication is performed using one Adv. SIMD multiplication in-

struction, the total number of multiplication instructions is the same for A64 and Adv. SIMD implementations. Figure ?? illustrates this comparative discussion for 128×128-bit multi-precision multiplication. Each 64×64-bit multiplication result is implemented using a pair of multiplication instructions, i.e., `MUL` and `UMULH` in A64 assembly language; therefore, the entire multiplication requires 8 multiplication instructions. Similarly, 8 SIMD multiplication instructions are required to implement the same function using Adv. SIMD assembly.

Based on the above discussion, roughly speaking, n Adv. SIMD multiplication instructions take about $\frac{5n}{2} + \frac{6n}{2} = 5.5n$ clock cycles, while n A64 multiplications take $\frac{4n}{2} + \frac{6n}{2} = 5n$ clock cycles on Cortex-A57 processors. Therefore, we claim that unlike AArch32 NEON instruction sets, AArch64 Adv. SIMD vectorization does not provide any performance improvement over A64 general-purpose registers for field arithmetic implementation. Based on this conclusion, we implement field arithmetic using A64 assembly instruction set, taking advantage of its wide 64-bit registers.

4.2 Finite Field Multiplication

The dominant field operation in any projective implementation is field multiplication and reduction, since all modular inversions are replaced with multiple multiplications. Therefore, optimized projective implementation requires efficient modular multiplication. We implement field multiplication using product scanning method for both $\mathbb{F}_{p764}$ and $\mathbb{F}_{p1014}$ fields using A64 instruction sets. We have access to 31×64-bit general registers and we are able to implement an optimized compact field multiplication function with a few number of data transfers between memory and registers.

4.3 Finite Field Reduction

Since we perform isogeny computations and point multiplications on Montgomery curves using Montgomery arithmetic, we use the efficient version of Montgomery reduction for our smooth primes as it is discussed in [?,?]. We remark that although the shape of our primes is slightly different compared to the SIDH smooth primes, we still can adopt the same optimization for our modular reduction implementation and achieve remarkable performance improvement compared to generic Montgomery or Barrett reduction. Thus, we implement customized Comba-based Montgomery reduction for each of the proposed primes, taking advantage of simplified formulas in [?], i.e., the reduction over $\hat{p} = p + 1$ which eliminates several single-precision multiplications by "0" limbs. In particular, $p764 + 1$ and $p1014 + 1$ have three and five 64-bit words equal to "0" in the lower half. Since we choose the primes with larger values of e_A, the total number of these zero limbs is the most possible value for each prime size. However, we note that since the chain of "0" in SIUS primes is relatively shorter than SIDH primes due to the smaller value of e_A for the same prime size, the overall performance of modular reduction is depreciated.

Table 3: Performance results ($\times 10^6$ CPU clock cycles) of SIUS protocol on various platforms. The verifier's confirmation and disavowal computations are implemented based on the protocol parameter b, while signer's operations are independent of this value.

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		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		8,052	10,957	20,913	23,453	8,868

4.4 Finite Field Inversion

We implement field inversion using Fermat's little theorem (FLT) with fixed window-based addition chain. Although FLT method is much slower than other non-constant time modular inversion algorithms such as Extended Euclidean Algorithm or Kaliski's inverse method in [**?**], since the total number of modular inversions is scarce in our protocol, we prioritize security over a small amount of performance improvement in using these algorithms. We implement modular inversion by using fixed 6-bit window addition chain method. We remark that constructing addition chains for the SIUS primes is different from the SIDH primes and using more efficient method of computing addition chain, such as hybrid-window method which is discussed in [**?**], yields negligible improvement in performance due to the shorter chain of "1" in the lower half of the prime.

5 Implementation Results and Discussion

Since this work is the first empirical implementation of a quantum-resistant undeniable signature, and the only other quantum-resistant undeniable signature [**?**] does not provide any performance results, we provide the performance measurements on a variety of platforms: a 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,

a 2.3 GHz NVIDIA Jetson TK1 equipped with a 32-bit ARMv7 Cortex-A15 running Ubuntu 14.04 LTS, and a desktop PC with a 2.1 GHz Intel x64 i7-6700 running Ubuntu 16.04 LTS. We also include our efficient results on ARMv8 processors to compare the efficiency of our optimized library. The binaries are compiled using `-O3 -fomit-frame-pointer -march=native` flags. Table ?? includes benchmark results of our SIUS implementation for both proposed quantum security levels. Results represent the average of 10^4 iterations reported in CPU clock cycles to provide a fair comparison of the performance on different platforms. Note that the verifier's disavowal computations differ in terms of the protocol value b, while signer's confirmation and disavowal computations stay the same.

The implementation results show that our hand-optimized library is almost 4.8X and 5.2X faster than generic implementation on the high-performance Cortex-A57 core over $p764$ and $p1014$, respectively. However, on the power-efficient Cortex-A53 core, the improvement factor is less and shows a speedup of 3.9X over $p764$ and 3.6X over $p1014$. We remark that our generic C finite field library is implemented in pure C without utilizing any multi-precision arithmetic libraries such as GMP[4] which implies that the more efficient generic implementation can be developed based on these libraries with the cost of extra dependencies during the compilation procedure.

The performance results on Jetson TK1 board with a high-performance 32-bit Cortex-A15 core is almost 3X slower than Cortex-A57 for the same implementation. It is because 64-bit platforms perform multi-precision arithmetic roughly twice as fast as 32-bit platforms. Moreover, the total number of available general registers in the ARMv8 processors is more compared to ARMv7-A which provides faster and much compact arithmetic with less number of data transfer to memory.

6 Conclusion

We have presented a constant-time software for supersingular isogeny-based undeniable signature protocol providing two different quantum security levels. We have built optimized libraries targeting the ARMv8-A family of processors using A64 assembly instruction set to achieve a factor speedup of up to 5.2X on a high-performance Cortex-A57 core. Moreover, taking advantage of reduced curve coefficient technique, we have decreased 25% of the SIUS signature and public-key sizes compared to its original scheme. To the best of our knowledge, this work is the first practical implementation of any quantum-resistant undeniable signatures found in the literature. We remark that since isogeny-based cryptosystems are younger than other post-quantum cryptography candidates, their performance and security are still required to be studied widely. For instance, developing more efficient formulas for isogeny computations will result in remarkable performance improvement of the overall protocol. Nevertheless, the signature size and performance of our software demonstrate the strong potential

[4] The GNU Multiple Precision Arithmetic Library

of this scheme as a quantum-resistant undeniable signature candidate. We hope that this work would be a paradigm shift towards motivating more investigation in this area.

7 Acknowledgement

The Authors would like to thank the anonymous SAC reviewers for their constructive comments. This work was supported by NSF grant no. CNS-1661557 and NIST award 60NANB16D246.