IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 20, 2019, accepted August 19, 2019, date of publication August 30, 2019, date of current version September 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2938682

An Efficient Signature Scheme From Supersingular Elliptic Curve Isogenies

YAN HUANG[1], FANGGUO ZHANG[2,3], ZHIJIE LIU[2], AND HUANG ZHANG[2]

[1]School of Electronic and Information Engineering, Sun Yat-sen University, Guangzhou 510006, China
[2]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China
[3]Guangdong Key Laboratory of Information Security, Guangzhou 510006, China

Corresponding author: Fangguo Zhang (isszhfg@mail.sysu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61672550, and in part by the National Key Research and Development Program of China under Grant 2017YFB0802503.

ABSTRACT Since supersingular elliptic curve isogenies are one of the several candidate sources of hardness for building post-quantum cryptographic primitives, the research of efficient signature schemes based on them is still a hot topic. In this paper, we present a many-time signature scheme based on the hash function from supersingular elliptic curve isogenies over the finite field $\mathbb{F}_{p^2}$ where $p = 2^{521} - 1$. Our signature scheme achieves smaller signature sizes relative to other post-quantum signature schemes based on supersingular elliptic curve isogenies, such as Galbraith's signature schemes (AsiaCrypt 2017) and Yoo's scheme (FC 2017). The structure of our scheme follows that of the hash-based signature scheme submitted to National Institute of Standards and Technology for post-quantum cryptography in 2018 with some modifications. To complete the construction, we firstly apply the method of Weil restriction to improve the efficiency of hash function from supersingular elliptic curve isogenies by approximately 30%, then propose a new Winternitz one-time signature scheme based on the hash function. Finally, we implement the signature scheme.

INDEX TERMS Elliptic curve isogenies, post-quantum cryptography, signature scheme.

I. INTRODUCTION

A recent research area for post-quantum cryptography is from supersingular elliptic curve isogenies [1]. These cryptosystems [2]–[4] are based on the difficulty of finding a path in the isogeny graphs of supersingular elliptic curves. Since the only known quantum algorithm for the problem has exponential complexity [5], it may be suitable for building post-quantum cryptography.

For the study of signature schemes based on supersingular elliptic curve isogenies, building a secure and efficient signature scheme is still a hot topic. Since Jao and De [2] proposed a key exchange protocol based on supersingular isogenies, some signature schemes based on it emerged one after another. Jao and Soukharev [6] gave an undeniable signature, Sun *et al.* [7] presented a designated verifier signature, Yoo *et al.* [4] designed a quantum-resistant signature and Galbraith *et al.* [8] constructed two signatures DFJP+FS and DFJP+U which were against classical and quantum adversaries, respectively. According to [8], it is shown that the protocol [2] might be dangerous in certain contexts, since it used small isogeny degrees and revealed auxiliary points. Hence, Galbraith *et al.* [8] provided another two schemes Sec3+FS and Sec3+U which both relied on the difficulty of computing the endomorphism ring of a supersingular elliptic curve. This difficulty has heuristic classical complexity of $\widetilde{O}(p^{1/2})$ bit operations, and quantum complexity $\widetilde{O}(p^{1/4})$ bit operations.

Since Castryck *et al.* [8] proposed a commutative key exchange protocol based on the class group from supersingular isogenies, De Feo and Galbraith [10] provided a new signature scheme based on the protocol. Deru *et al.* [11] speeded up the signature scheme. By relying on the programmes proposed by Kleinjung [12] and working over the maximal order, Beullens *et al.* [13] found the generator of the class group and constructed a new signature scheme, which was 300 times faster than the optimized version [11]. Note that Bonnetain and Schrottenloher [14] and Peikert [15] reassessed the security of the key exchange based on the class group and found that the CSIDH failed to achieve 64 bits of quantum security.

Up to now, these signature schemes mentioned above are constructed from identification protocols by using the

The associate editor coordinating the review of this manuscript and approving it for publication was Tony Thomas.

Fiat-Shamir transform [16] against classical adversaries, or Unruh transform [17] against quantum adversaries, respectively. In these identification protocols, the prover and verifier need to interact at least λ times where λ is the security parameter. As the isogeny does not have rich algebraic structure and the interactive process takes one bit at a time, the efficiency of the corresponding signature schemes is relatively slow.

It is well known that one signature scheme can be constructed not only based on identification protocols, but also based on cryptographic hash functions which combines with tree structures.

For the hash-based digital signature schemes, there are two schemes submitted to National Institute of Standards and Technology [18], [19] in response to their call for post-quantum cryptography standardization. These schemes are based on hash functions such as SHA-2, SHA-3, etc. If these hash functions can be replaced by ones which are provable secure under the assumptions of computationally hard problems such as those from supersingular elliptic curve isogenies, the security of these schemes would be increased.

In 2009, Charles *et al.* [20] proposed an expander hash (CGL), which is based on the isogeny graphs of supersingular elliptic curves over finite fields. According to [21, Proposition 15], the CGL hash suffered from a collision attack when the initial curve is special. Namely, its endomorphism ring is known [21, Proposition 15]. Doliskani *et al.* [20] proposed a variant of the expander hash (JGP) that was more efficient than the original CGL algorithm. Besides, in order to prevent this attack, the starting elliptic curve is chosen randomly such that the endomorphism ring computation problem is hard [21].

A. OUR CONTRIBUTIONS

In this paper, we firstly optimize the hash function from supersingular elliptic curve isogenies by taking advantage of the Weil restriction. This trick can transform all the arithmetic operations over $\mathbb{F}_{p^2}$ into those over $\mathbb{F}_p$, which speeds up the computation of hash function by about 30%. Then we propose a Winternitz one-time signature scheme based on the hash function and utilize the structure of hash-based signature to change the one-time signature into many-time signature. The scheme has shorter signature size than other post-quantum signature schemes based on supersingular elliptic curve isogenies. Finally, we implement the signature scheme.

B. ORGANISATION

The rest of the paper is organized as follows. We shall briefly introduce the preliminaries in Section II. In Section III, we optimize the hash function based on supersingular elliptic curve isogenies, propose a keyed one-way function based on the hash function and construct a new Winternitz one-time signature scheme based on the keyed one-way function. A many-time signature scheme based on hash functions from the supersingular elliptic curve isogenies is presented in Section IV. We give the efficiency analysis in Section V and a brief conclusion in Section VI.

II. PRELIMINARIES

This section sets the stage by reviewing some background about supersingular elliptic curve isogenies and Kummer surfaces, keyed one-way function [23] and signature schemes based on hash functions [19].

A. SUPERSINGULAR ELLIPTIC CURVE ISOGENIES AND KUMMER SURFACES

We summarize the required background about isogenies according to the theory in [24]. Let E, E' be two elliptic curves over a finite field $\mathbb{F}_q$. An isogeny ψ is a non-constant morphism $E \to E'$ of elliptic curves that preserves the group structure. The degree of an isogeny ψ is the degree of ψ as a morphism. An isogeny of degree l is called an l-isogeny. If ψ is separable, then $\deg\psi = \sharp\ker\psi$. If there is a separable isogeny between two curves, we say that they are isogenous. Tate's theorem is that two curves E, E' over $\mathbb{F}_q$ are isogenious if and only if $\sharp E(\mathbb{F}_q) = \sharp E'(\mathbb{F}_q)$. A separable isogeny can be identified with its kernel [25]. Given a subgroup G of E, we can use Vélu's formulae [26] to explicitly obtain an isogeny $\psi : E \to E'$ with kernel G such that $E' \cong E/G$. Given a prime l, the torsion group $E[l]$ contains exactly $l+1$ cyclic subgroups of order l, each of which corresponds to a different isogeny.

An isogeny $\psi : E \to E'$ such that $E = E'$ is called an endomorphism. The set of endomorphisms of an elliptic curve, denoted by $End(E)$, has a ring structure which is either an order in a quadratic imaginary field or a maximal order in a quaternion algebra. In the first case, we say that the curve is ordinary, whereas in the second case we say that the curve is supersingular.

It can be shown that every supersingular elliptic curve can be defined over $\mathbb{F}_{p^2}$, thus its j-invariant is over $\mathbb{F}_{p^2}$. Although there are $N_p := \lfloor \frac{p}{12} \rfloor + \epsilon_p$ supersingular j-invariants, with $\epsilon_p = 0, 1, 1, 2$ when $p = 1, 5, 7, 11 \mod 12$ respectively, we are not aware of an efficient method to encode each of these j-invariants into $\log_2 p$ bits, so we represent a supersingular j-invariant with $2\log_2 p$ bits. For any prime $l \neq p$, one can construct a so-called isogeny graph, where each vertex is associated to a supersingular j-invariant, and an edge between two vertices is associated to an l-isogeny between the corresponding vertices. Isogeny graphs are regular with degree $l+1$; they are undirected since any isogeny from j_1 to j_2 corresponds to a dual isogeny from j_2 to j_1. Isogeny graphs are also very good expander graphs G_l [27]. Next, we give three hard problems in G_l according to [22]. Let $n = \log_2 p$. For a prime $l \neq p$, denote by G_l the graph of supersingular elliptic curves over $\mathbb{F}_{p^2}$.

Problem 1: Find curves $E_1, E_2 \in G_l$ and two distinct isogenies $\phi_1, \phi_2 : E_1 \to E_2$ of degrees l^{rn} and l^{sn} for some integers $r, s > 0$.

Problem 2: Given a curve $E \in G_l$, find an endomorphism "$\phi \in End(E)/\mathbb{Z}$" of degree l^{rn} for some even $r > 0$.

Problem 3: Given curves $E_1, E_2 \in G_l$, find an isogeny $\phi : E_1 \to E_2$ of degree l^{rn} for some integer $r > 0$.

In [22], it had been shown that **Problem 1** is equivalent to **Problem 2** in G_l, while finding a solution to **Problem 3** implies a solution to **Problem 1**. Due to Biasse *et al.* [5], **Problem 3** has heuristic classical complexity of $\widetilde{O}(p^{\frac{1}{2}})$ bit operations, and quantum complexity $\widetilde{O}(p^{\frac{1}{4}})$.

For a genus-2 curve C over $\mathbb{F}_q$, J_C is its Jacobian group, the quotient $J_C/\{\pm 1\}$ is called Kummer surface. There exists fast Kummer surface K^{Sqr}, which has efficient arithmetic operations and isogeny computations. By leveraging a linear projective isomorphism between $J_C/\{\pm 1\}$ and K^{Sqr} according to [28], all the arithmetic operations on $J_C/\{\pm 1\}$ can be transformed into those on K^{Sqr}. For the evaluation of isogenies, Costello [31] presents a fast method to optimize the (2, 2)-isogenies over $\mathbb{F}_p$ and apply the Weil restriction to transfer the evaluation of isogenies on Montgomery curves over $\mathbb{F}_{p^2}$ into the evaluation on Kummer surfaces K^{Sqr} over $\mathbb{F}_p$.

B. KEY ONE-WAYNESS

Keyed One-Way Function: A keyed one-way function family $\mathcal{F}_n = \{F_k : \{0,1\}^n \to \{0,1\}^n | k \in \{0,1\}^n\}$ are parameterized by a key $k \in \{0,1\}^n$ and the security parameter n [35]. The key one-wayness of the function family can be defined as follows:

Definition 1 (Key one-wayness(KOW) [23]): Let $\mathcal{F}_n$ be a family of keyed one-way functions as above. We call $\mathcal{F}_n$ is (t, ϵ) key one-wayness, if the success probability

$$Adv_{\mathcal{A}}^{KOW} = Pr[(x,k) \xleftarrow{\$} \{0,1\}^n \times \{0,1\}^n, y \leftarrow f_k(x), k' \leftarrow \mathcal{A}(x,y) : y = f_{k'}(x)]$$

of any adversary $\mathcal{A}$ that runs in time t is at most ϵ.

C. SIGNATURES

A signature scheme is a tuple of probabilistic polynomial-time algorithms (Gen, Sign, Verify) [35]. We use the standard security notion of existentially unforgeable under adaptively chosen messages attacks (EU-ACMA) [36], which is defined using a game between a challenger and a forger. Namely, a forger can ask a signing oracle $Sign(sk, .)$ for polynomial many signatures of messages of his choice. Then a successful attack is considered if the forger is able to produce a valid pair of message and signature for a message different from those queried to the oracle. The specific process is as follows:

Definition 2 (EU-ACMA): Setup: The challenger runs Gen to output a pair of keys (pk, sk), and give the public key pk to the forger.

Queries: The forger $\mathcal{A}$ adaptively requests at most q_s messages $M_1, ..., M_{q_s}$. The challenger responds to the *ith* query with a valid signature σ_i where $i \in \{1, ..., q_s\}$.

Output: Finally, the forger $\mathcal{A}$ outputs a valid message-signature pair (M^*, σ^*) and wins the game if $M^* \notin M_i$ for all $i \in \{1, ..., q_s\}$.

The advantage of the forger $\mathcal{A}$ is defined as

$$Adv_{\mathcal{A}}^{EU-ACMA} = Pr\left[\begin{array}{c}(sk, pk) \leftarrow Gen(1^n); \\ (M^*, \sigma^*) \leftarrow \mathcal{A}^{Sign(sk,.)}(pk) : \\ M^* \notin M \wedge Verify_{pk}(M^*, \sigma^*) = True\end{array}\right].$$

The probability is taken over the coin tosses of the Gen, Sign and $\mathcal{A}$.

The signature is (t, ε, q)-EU-ACMA if there is no t-time adversary that succeeds with probability $\geq \varepsilon$ after making $\leq q$ signature oracle queries.

A $(t, \varepsilon, 1)$-EU-ACMA secure signature scheme is called a one-time signature scheme that is existentially unforgeable under the 1-adaptively chosen message attack.

D. HASH-BASED SIGNATURE SCHEMES

In this subsection, we mainly review the hash-based signature scheme submitted to NIST for post-quantum cryptography in 2018 [19]. The scheme combines some basic blocks such as a Winternitz one-time signature scheme, a subsets-based scheme, secret key caching, as well as batch signing with a hyper-tree construction, which achieves the goal of signing many messages per key pair.

1) WINTERNITZ ONE-TIME SIGNATURE SCHEME (WOTS):

The WOTS uses one string of the signing key to sign several bits of the message digest simultaneously. The key point is to iteratively use a one-way function $F : \{0,1\}^n \to \{0,1\}^n$ on a secret input. For a message digest M of n bits, one firstly chooses parameters l_1 and w such that $l_1 \cdot \log_2 w = n$, then decomposes M into l_1 chunks of $\log_2 w$ bits $(x_1, ..., x_{l_1})$. Secondly, he computes $C = \sum_{i=1}^{l_1}(w - x_i)$, and decomposes C into l_2 chunks $(c_1, ..., c_{l_2})$, each of which is of $\log_2 w$ bits, and appends $(c_1, ..., c_{l_2})$ to x to form $b = (x_1, ..., x_{l_1}, c_1, ..., c_{l_2}) = (b_1, ..., b_l)$ where $l = l_1 + l_2$. The secret key consists of l n-bits strings $(s_1, ..., s_l)$ and the public key is $(F^{w-1}(s_i))_{1 \leq i \leq l}$. The signer issues $(y_i)_{1 \leq i \leq l} = (F^{b_i}(s_i))_{1 \leq i \leq l}$ as the one-time signature of M. Verification is done by computing $F^{w-1-b_i}(y_i)$ and comparing the result with the public key element $F^{w-1}(s_i)$ for i from 1 to l. In order to reduce the size of public key, the L-tree construction is used. It uses these public keys as leaves, and generates the internal nodes with a hash function. Nevertheless, if the number of a level in this tree is an odd, the rightmost node is lifted up one level. The root of the tree is then used as the public key pk. We call the invariant WOTS+. The security of the WOTS+ mainly relies on the one-wayness of F and collision-resistance of hash functions.

2) PRNG TO OBTAIN A RANDOM SUBSET FROM A TREE (PORST)

The signature constructed by the PORST is a few-time signature scheme, it uses a pseudo-random function to obtain a random subset from a key set which acts as the leaves of a tree. The secret key is n-bits strings $(s_1, ..., s_t)$. The public key pk is the root of a Merkle tree which uses these t values as leaves and uses one hash function to compute internal nodes.

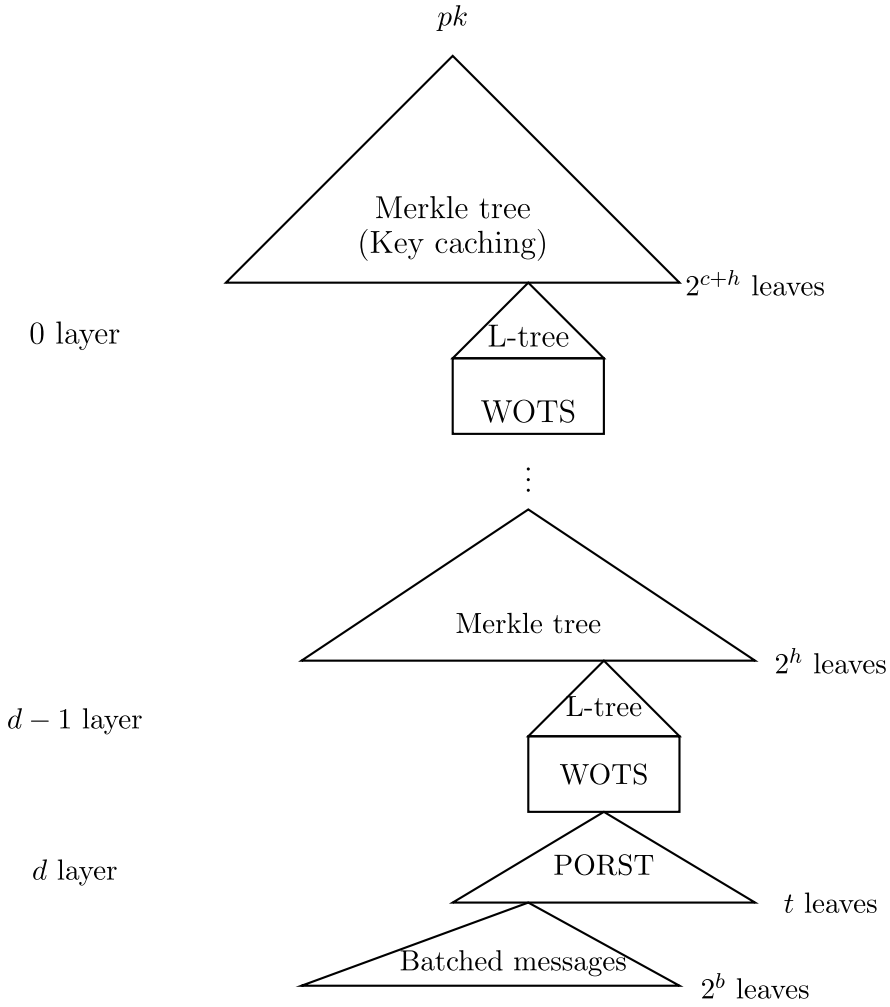


FIGURE 1. The structure of hash-based signature scheme.

For a message digest m of n bits, using a pseudo-random function G, the signer takes m and an address parameter A as input, outputs k elements from $(s_1, ..., s_t)$ as a part of the signature. Meanwhile, the signer computes an optimal authentication path for the k elements in the Merkle tree as the other part of the signature. Verification is done by computing the root of the Merkle tree and comparing the result with the public key. The security of the PORST signature scheme mainly relies on the hardness of the subset-resilience problem and the collision resistance of the hash function.

3) GRAVITY-SPHINCS

Gravity-SPHINCS [19] is a many-time signature based on hash functions. It includes four parts: key caching of height c; d layers of WOTS+ instances (each of which includes a Merkle tree of height h and a WOTS+); PORST signature and batch messages of height b at the bottom of the hyper-tree. This structure is shown in FIGURE.1

The process is taken as follows. Firstly, the secret keys are two n-bits values *seed* and *salt*, the *seed* is used for pseudorandom key generation and the *salt* is used to generate an unpredictable index and pseudo-random values to randomize the message hash. The public key *pk* is the root of a Merkle tree of height $h + c$ whose leaves are WOTS+ public keys. The first step of the signature is to gather all the messages and to compute a Merkle tree from their respective hashes by applying the batch signing, then the root m of the Merkle tree is the message signed by PORST signature. The authentication path A_{batch} of the corresponding message M_e, as well as its index e (i.e. the index of the message required signed) are as part of the Gravity-SPHINCS signature. The second step is the PORST signature, which uses the secret key *seed* and an address A to generate t values as the secret keys of the PORST signature, and the signature of m is (σ_d, oct). The public key of PORST signature is regarded as the messages for the next signature. The third step repeats WOTS+ and constructs Merkle trees for d times, the secret keys for WOTS+ are generated by the secret key *seed* and an address A. Then the d time signatures are $(\sigma_{d-1}, Auth_{d-1}, ..., \sigma_0, Auth_0)$. At the last step, we need to compute 2^{c+h} WOTS+ public keys

and the authentication path $Auth_{h+c}$ until we reach the root of the hyper-tree. The $Auth_c$ is to remove the first h nodes of the authentication path $Auth_{h+c}$. Then the signature is $\sigma = (s, i, b_i, \sigma_d, oct, \sigma_{d-1}, Auth_{d-1}, ..., \sigma_0, Auth_0, Auth_c)$. Given a signature σ' and the public key pk, the verification process consists of recomputing the root of the Merkle tree for batch messages, the root of the Merkle tree for PORST signature, the roots of the Merkle trees for WOTS+ for d times until reaching the root of the hyper-tree. If all verifications succeed and the root of the top tree equals pk, then the signature is accepted. The security of Gravity-SPHINCS can be proved by using the product composition [37] which combines the security of each part together.

III. HASH FUNCTION AND KEYED ONE-WAY FUNCTION FROM SUPERSINGULAR ELLIPTIC CURVE ISOGENIES AND WINTERNITZ ONE-TIME SIGNATURE

In this section, we optimize the hash function and present a keyed one-way function based on hash function from supersingular isogenies. Basing on these two function, we propose a Winternitz one-time signature and prove its security.

A. HASH FUNCTION AND KEYED ONE-WAY FUNCTION FROM SUPERSINGULAR ISOGENIES

Doliskani *et al.* [22] proposed a hash function $H(E_0, m)$ in Algorithm 2 from supersingular elliptic curve isogenies. The function takes an initial curve E_0 and arbitrary length message m as input and is constructed by leveraging the Merkle-Damgård structure [29]. The compression function $h(E, m, c)$ in Algorithm 2 is called by taking a supersingular elliptic curve E, an $n-$bit message m and an index c as inputs, computing 2^n-isogenies and outputing a supersingular elliptic curve E'.

We utilize the trick of the Weil restriction [31] to optimize the compression function. Specifically, in Algorithm 1, Step 1 obtains two independent points of $E[2^n]$ as generators canonically according to the input index c and a fixed table $T1$ (or $T2$). Step 2 computes the kernel generator point R of an isogeny. These two steps are the same as those in [22]. Step 3 and Step 4 convert the supersingular elliptic curve E and the kernel point R on E over $\mathbb{F}_{p^2}$ into the Kummer surface K^{Sqr} and the corresponding point $R_{K^{Sqr}}$ on K^{Sqr} over $\mathbb{F}_p$. They make use of Scholten's construction [30] to change the Montgomery curve E into the general Kummer surface J_C, and then leverage the isomorphism map mentioned in [32] to convert the general Kummer surface J_C into fast Kummer surface K^{Sqr}. Thus an isogeny ϕ can be evaluated over a prime field $\mathbb{F}_p$ at Step 5, which is similar to the approach in [33]. Then Step 6 converts the Kummer surface K^{Sqr} back to the Montgomery curve E'. These operations enhance the efficiency of the compression function. The details of Step 3, 4 and 5 are presented in Appendix A. Now we analyze the efficiency. Let m, s denote the multiplication and squaring operations over $\mathbb{F}_p$ and M, S, I denote the multiplication, squaring and inversion operations over $\mathbb{F}_{p^2}$. According to the implementation results, $m = 155$ cycles, and $s = 105$ cycles. In light of common approximations, $M \approx 3m$, $S \approx 2s + m$ and $I \approx 100M$, a square root in $\mathbb{F}_{p^2}$ requires $(2\log_2 p+1)M + 1S + 1I$.

Algorithm 1 $h(E, m, c)$

Input : A supersingular curve E, an $n-$bit message m, An integer c.
Output : A curve E'
1: Obtain generators P, Q of $E[2^n]$ deterministically from $T1[c]$ (or $T2[c]$)
2: Compute $R = P + mQ$
3: Convert E over $\mathbb{F}_{p^2}$ into Kummer surface K^{Sqr} over $\mathbb{F}_p$
4: Map the point R on E into the point $R_{K^{Sqr}}$ on K^{Sqr}
5: Compute an isogeny $\phi : K^{Sqr} \rightarrow \widehat{K^{Sqr}}$ with kernel $\langle R_{K^{Sqr}} \rangle$
6: Convert $\widehat{K^{Sqr}}$ over $\mathbb{F}_p$ into E' over $\mathbb{F}_{p^2}$
7: Return E'

TABLE 1 presents the efficiency analyses of the compression functions in Algorithm 1 and in [22]. By comparison, our function has been speeded up by 30%, thus the hash function in Algorithm 2 can be further improved by 30% on average.

Algorithm 2 $H(E_0, m)$

Input : A supersingular curve E_0, a message m
Output : A curve E_2
1: Pad the message m to get $m = m_0||m_1||...||m_k$, where each block m_i is n bits
2: $c = 0$
3: $E_1 := E_0, E_2 := E_0, E_3 := E_0$
4: for $i = 0$ to k
5: do {
6: Compute the isogeny $E_3 := h(E_2, m_i, c)$
7: $c := c + 1$
8: while $(E_3 = E_1)$
9: $E_1 := E_2, E_2 := E_3$
10: $c := c + 1$ }
11: return E_2

The hash function $H(E_0, m)$ is also proved to be preimage-resistant and collision-resistant when the initial curve E_0 is chosen randomly.

Theorem 1 (Preimage Resistance (PR) [22]): If there is an efficient algorithm for finding preimages in the hash family, then there is an efficient algorithm for **Problem 3**.

Theorem 2 (Collision Resistance (CR) [22]): If there is an efficient algorithm for finding collisions in the hash family, then there is an efficient algorithm for **Problem 1** and **Problem 2**.

TABLE 1. The efficiency comparison of computing the compression function by utilizing chained 2-isogenies on Montgomery curves over $\mathbb{F}_{p^2}$ [22] with chained (2,2)-isogenies on Kummer surfaces over $\mathbb{F}_p$ [31].

	The computation in [22]			Algorithm 1		
operations	M	S	cycles	m	s	cycles
1. Generators P and Q	1496.3	7	697845	-	-	697845
2. Compute $R = P + mQ$	3126	2084	2422650	-	-	2422650
3. Convert E to K^{Sqr}	0	0	0	17688	42	2746050
4.Map the point on E into the point K^{Sqr}	0	0	0	738	10965	1265715
$5.2^{521}-$isogeny	543 920	271960	561337920	1095663	1087848	377795600
6. Convert K^{Sqr} into E	0	0	0	4358	64	682210
Total cost	-	-	564,458,415	-	-	385,610,070

Keyed One-Way Function Based on the Hash Function: The keyed one-way function is a keyed hash function in essence. If there exists a hash function, it is easy to construct a keyed hash function [35]. Therefore, our keyed one-way function H_{kow} can be defined as $H_k(E_0, x) = H(E_0, k||x)$, i.e., computing a hash function based on supersingular isogeny with the secret key k as part of the input. The key one-wayness can be reduced to the preimage resistance of the hash function based on supersingular isogeny.

B. WINTERNITZ ONE-TIME SIGNATURE (WOTS)

The Winternitz one-time signature based on the keyed one-way function is a variant of WOTS [23]. Now, we use a keyed hash function from supersingular elliptic curve isogenies H_{kow} to specify it with some modifications.

1) KEY GENERATION

Firstly, we choose the Winternitz parameter $w = 2^r \in \mathbb{N}$, $t \geq 1$, to define the compression level. Next we choose a random $x \xleftarrow{\$} \{0,1\}^n$ and a random supersingular elliptic curve $E_0 \in \{0,1\}^{2n}$. The signature secret key consists of l bit strings of length n chosen uniformly at random $sk = (sk_1, ..., sk_l) \xleftarrow{\$} \{0,1\}^{\{n \cdot l\}}$, where $l = l_1 + l_2$, $l_1 = \lceil n/r \rceil$ and $l_2 = \lceil (\lceil \log_2 n - \log_2 r \rceil + r)/r \rceil$.

Every element of the public key is computed by iteratively using the keyed hash function from supersingular elliptic curve isogenies $H(E_0, k||x)$ for $w-1$ times, where the output of the i iteration is the key of the $i+1$ iteration, i.e., $k_{i+1} = H(E_0, k_i||x)$. Therefore, the public key can be computed by

$$\begin{aligned} pk &= (pk_0, pk_1, ..., pk_l) \\ &= (x, H^{w-1}(E_0, sk_1||x), ..., H^{w-1}(E_0, sk_l||x)). \end{aligned}$$

When constructing an $L-$tree as in Section II.D, the leaves are $pk_1, ..., pk_l$. Let the root node of this $L-$tree be E_0', the public key could be redefined as $pk = (E_0, E_0', x)$.

2) SIGN

We describe how to sign an $m-$bit message $M = (M_1, ..., M_{l_1})$ given in base-w representation, i.e., $M_i \in \{0, ..., w-1\}$ for $i = 1, ..., l_1$. We begin by computing the checksum $C = \sum_{i=0}^{l_1}(w - 1 - M_i)$ and representing it to base w as $C = (C_1, ..., C_{l_2})$. The length of the base-w representation of C is at most l_2 since $C \leq l_1(w-1)$. Then we set $B = (b_1, b_2, ..., b_l) = M||C$ (we can use the function $checksumed(M)$ to describe the process later). The signature of message M is computed as

$$\sigma = (\sigma_1, ..., \sigma_l) = (H^{b_1}(E_0, sk_1||x), ..., H^{b_l}(E_0, sk_l||x)).$$

3) VERIFY

Given a message M and signature $\sigma = (\sigma_1, ..., \sigma_l)$, the verifier first computes the base-w string $B = (b_1, b_2, ..., b_l)$ as described above. Then he

1. computes

$$\begin{aligned} (pk_1, pk_2, ..., pk_l) = (&H^{w-1-b_1}(E_0, \sigma_1||pk_0), ..., \\ &H^{w-1-b_l}(E_0, \sigma_l||pk_0)), \end{aligned}$$

2. recomputes the root pk' of the $L-tree$ and checks $pk' \stackrel{?}{=} E'$.

Now, we give the security analyses of the WOTS, as well as the WOTS+ which uses a so-called $L-$tree to compress the l values of the WOTS public key into a single $2n-$bit string.

Theorem 3: If H_{kow} is a $(t_{KOW}, Insec^{KOW}(H_{kow}))$ keyed one-way function, then WOTS is existentially unforgeable with

$$Insec^{EU-ACMA}(WOTS(H_{kow})) \leq l^2 w^2 \mathrm{k}^{w-1} Insec^{KOW}(H_{kow})$$

where l and w are defined above and the upper bound k is denoted as follows: for each pair (E_0, x), there exist at most $\mathrm{k}-1$ different curves $k_1, ..., k_{\mathrm{k}-1}$ such that $H(E_0, k_i||x) = H(E_0, k||x)$ for $i = 1, ..., \mathrm{k}-1$.

Proof 1: Suppose there exits an adversary A attacking the WOTS, then we can use the adversary A to construct an adversary B against the key one-wayness of H_{kow}. The signing oracle **Sign** is simulated by the adversary

The adversary B aims at producing a key k' such that $H(E_0, k'||x) = y$ for x, y provided as input. The adversary B first generates a regular WOTS signature key pair and chooses random positions α and β (Line 1, 2). Then he computes the WOTS public key using value x. The public key pk_α

Algorithm B

Input : Security parameters n, Winternitz parameter w, initial curve E_0, description of H_{kow}, key one-wayness challenge (x, y) as in Definition 2.1,
Output : k', such that $H(E_0, k'||x) = y$
1: Generate a WOTS signature key sk
2: Choose indices $\alpha \in \{1, ..., l\}$,$\beta \in \{1, ..., w-1\}$ uniformly at random
3: Compute the public key $pk_0 = x$,
$pk_i = H^{w-1}(E_0, sk_i||x)$
for $i = 1, ..., l, i \neq \alpha$ and $pk_\alpha = H^{w-1-\beta}(E_0, y||x)$
4: Run the adversary A
5: When A queires **Sign** with message M, compute
$(b_1, ..., b_l) \leftarrow checksumed(M)$
6: If $b_\alpha < \beta$ return fail
7: Generate signature σ and respond to the adversary A
8: When the adversary A returns valid (σ', M') then compute $(b'_1, ..., b'_l) \leftarrow checksumed(M)$
9: If $b'_\alpha \geq \beta$ return fail
10: Compute $k' \leftarrow H^{\beta-1-b'_\alpha}(E_0, \sigma'_\alpha||x)$
11: If $H(E_0, k'||x) \neq y$ return fail
12: Return k'

is computed by inserting y at position β in the hash chain (Line 3). Next, the adversary B calls the forger and waits for it to ask an oracle query (Line 4, 5). The forger's query can only be answered if $b_\alpha \geq \beta$ holds, because B doesn't know the first β entries in the corresponding hash chain (Line 6). The forgery produced by the forged signature is only meaningful to B if $b'_\alpha < \beta$ holds (Line 9). Only then the σ'_α in the forged signature might yield a key k' such that $H(E_0, k'||x) = y$ holds.

We now compute the success probability of B. Assume the forger queries the signing oracle. The probability of $b_\alpha \geq \beta$ in Line 6 is at least $(lw)^{-1}$. This is due to the checksum which guarantees that not all of the b_i are zero simultaneously. The probability in Line 8 that the forger succeeds is at least $Insec^{EU-CMA}$ by definition. The probability holds under the condition that the public key computed in Line 3 resembles a regular public key which is the case if there exists a key k such that $H^\beta(E_0, k||x) = y$. This happens with probability at least $1/\mathrm{k}^\beta$. The probability of $b'_\alpha < \beta$ in Line 9 is at least $(lw)^{-1}$. The probability that $y = H(E_0, k'||x)$ holds in Line 11 is at least $1/\mathrm{k}^{w-1-\beta}$. This is because there exists k^{w-1} keys mapping x to pk_α after $w-1$ iterations and only k^β of these keys map x to y after β iterations. Therefore, we have

$$Insec^{KOW}(H_{kow}) \geq Insec^{EU-CMA}/(l^2w^2\mathrm{k}^\beta\mathrm{k}^{w-1-\beta}) = Insec^{EU-CMA}/(l^2w^2\mathrm{k}^{w-1}) \quad (1)$$

Theorem 4 [19]: Let H_{kow} and H_{cr} be depicted as above. We consider the following resources ξ : the time τ, the number of queries to the signature scheme q and the number of queries to H_{kow} and H_{cr} respectively $q_{H_{kow}}$ and $q_{H_{cr}}$. The unforgeability of the WOTS+ based on H_{kow} and H_{cr} can be bounded by the unforgeability of WOTS and the collision resistance of H_{cr}, i.e.,

$$Insec^{EU-ACMA}(WOTS + (H_{kow}, H_{cr}); \tau, \varepsilon, q, q_{H_{kow}}, q_{H_{cr}}) \leq l^2w^2\mathrm{k}^{w-1}Insec^{KOW}(H_{kow}; \tau, q, q_{H_{kow}}) + Insec^{CR}(H_{cr}; \tau', q''_{H_{cr}})$$

where l, w and k are defined above, $q = 1$, $q''_{H_{cr}} = q_{H_{kow}} + 2(l-1)$, $\tau = \tau + c(l-1)$ and $\tau' = \tau + c(w-1)l$ for some constant c.

IV. SIGNATURE SCHEMES

In this section, we exploit those constructions presented in Section III to give a new signature based on hash functions from supersingular elliptic curves isogenies. Firstly, we give basic parameters and basic functions used in our signature schemes. Secondly, exploiting these hash functions, we define some internal algorithms that are the building blocks of our signatures. Lastly, we propose a new signature scheme.

A. PUBLIC PARAMETERS

The signature scheme from supersingular isogenies requires some parameters which are also defined in the scheme [19].

- A prime $p = 2^n - 1$,
- a supersingular elliptic curve $E_0 : y^2 = x^3 + A_0x^2 + x$ where $A_0 \in \mathbb{F}_{p^2}$,
- the message space M, which is a subset of bit string $\{0, 1\}^*$,
- the batching height b, a non-negative integer,
- the PORS set size t, a positive power of two,
- the PORS subset size k, a positive integer such that $k \leq t$,
- the Winternitz depth w, a power of two such that $w \geq 2$,
- the Winternitz width $l = \mu + \lfloor \log_2 \mu(w-1)/\log_2 w \rfloor + 1$ where $\mu = \lceil n/\log_2 w \rceil$,
- the Merkle height h in each WOTS+ instance (i.e. WOTS+ with a Merkle tree), a non-negative integer,
- the layers d of WOTS+ instances, a non-negative integer,
- the key caching height c, a non-negative integer.

B. PRIMITIVES

Our signature scheme mainly uses the following three functions.

The collision-resistant hash function $H_{cr}(E_0, m) : \{0, 1\}^{2n} \times \{0, 1\}^* \rightarrow \{0, 1\}^{2n}$ takes as input a supersingular curve and an arbitrary length message m, and outputs a supersingular curve. This function is used for computing L-trees and Merkle trees, so the length of the message m in H_{cr} is always $4n$ bits.

A keyed one-wayness function $H_{kow}(E_0, s||x)$: $\{0, 1\}^{2n} \times \{0, 1\}^{2n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$, takes as input a supersingular curve, a secret key s and a random value x, and then outputs

TABLE 2. The functionalities, inputs and outputs of functions.

Functions	Inputs and outputs
Operation adddress	$make - addr : \{0, ..., d\} \times \mathbb{N} \rightarrow \mathrm{A}$
Operation adddress	$incr - addr : \mathrm{A} \times \mathbb{N} \rightarrow \mathrm{A}$
L-tree root	$L - tree : \{0,1\}^{2n} \times \{0,1\}^{2nl} \rightarrow \{0,1\}^{2n}$
Merkle root	$Merkle - root : \{0,1\}^{2n} \times \{0,1\}^{2n \times 2^h} \rightarrow \{0,1\}^{2n}$
Merkle Tree Authentication	$Merkle - auth_h \{0,1\}^{2n} \times \{0,1\}^{2n \cdot 2^h} \times \{0, ..., 2^h - 1\} \rightarrow \{0,1\}^{2nh}$
Merkle Tree Root Extraction	$Merkle - extract_h : \{0,1\}^{2n} \times \{0,1\}^{2n} \times \{0, ..., 2^h - 1\} \times \{0,1\}^{2nh} \rightarrow \{0,1\}^{2n}$
Octopus Authentication	$Octopus - auth_h : \{0,1\}^{2n} \times \{0,1\}^{2n \times 2^h} \times \{0, ..., 2^h - 1\}^k \rightarrow \{0,1\}^* \times \{0,1\}^{2n}$
Octopus Root Extraction	$Octopus - extract_{h,k} : \{0,1\}^{2n} \times \{0,1\}^{2nk} \times \{0,1\}^* \rightarrow \{0,1\}^{2n} \bigcup \{\perp\}$
PORS	$PORS : \{0,1\}^{2n} \times \{0,1\}^{2n} \rightarrow N \times T^k$

a supersingular curve. This function is used for computing WOTS+ signature.

A pseudorandom function $G(seed, a_i)$: $\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ takes as input a *seed* and an address a_i, outputs a secret key. This function is used for generating secret keys of WOTS+ and PORST signature. For example, we can take a variant of AES [34] as the pseudorandom function.

Remark: It is worth nothing that is the computation of hash function $H(E_0, x_1||x_2)$ for the construction of tree which takes an initial curve E_0, two nodes x_1 and x_2 represented hash values as input. Since x_1 and x_2 are also curves over $\mathbb{F}_{p^2} = \mathbb{F}_p/\langle x^2+1\rangle$, and can be represented as $x_1 = a_{11} + a_{12}i$ and $x_2 = a_{21} + a_{22}i$, we always regard $x_1||x_2$ as 4 message blocks $a_{11}||a_{12}||a_{21}||a_{22}$, where each block is n bits.

C. INTERNAL ALGORITHMS

In this part, we describe some internal algorithms with the same notation as in [19]. TABLE 2 presented some functions similar to those in [19], we put the explicit definitions in Appendix B. These functions described below are mainly used for Winternitz signature.

Winternitz Public Key Generation (The Function WOTS +Genpk): $\{0,1\}^{2n} \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^{2n}$ takes as input a supersingular curve E_0, a secret value *seed*, an address $\mathbf{A}$ and a random value x, and outputs the associated Winternitz public key p as follows:

- Let address $\mathbf{A}_i = incr - addr(\mathbf{A}, i-1)$ for $i = 1, .., l$ and pad $\mathbf{A}_i$ to n bits.
- $s_i = G(seed, \mathbf{A}_i)$ and pad s_i to n bits for $i = 1, ..., l$.
- Compute the public values $pk_i = H^{w-1}(E_0, s_i||x)$ for $i = 1, ..., l$.
- Compute $p \leftarrow L - tree(E_0, pk_1, ..., pk_l)$.

Winternitz Signature: The function WOTS+: $\{0,1\}^{2n} \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^{2nl}$ takes as input a supersingular curve E_0, a secret value *seed*, an address $\mathbf{A}$, a hash m and a random value x and outputs the associated Winternitz signature $\sigma \in \{0,1\}^{2nl}$ as described in Section III. B:

- Exploit the function $checksumed(m)$ to split the hash m into l blocks $b_1||b_2||...||b_l$.
- Let address $\mathbf{A}_i = incr - addr(\mathbf{A}, i-1)$ and pad $\mathbf{A}_i$ to n bits for $i = 1, .., l$.
- $s_i = G(seed, \mathbf{A}_i)$ for $i = 1, .., l$.
- $\sigma = (\sigma_1, ..., \sigma_l) = (H^{b_1}(E_0, s_1||x), ..., H^{b_l}(E_0, s_l||x))$.

Winternitz Public Key Extraction (The Function WOTS+ Extractpk):

$$\{0,1\}^{2n} \times \{0,1\}^{2n} \times \{0,1\}^{2nl} \times \{0,1\}^n \rightarrow \{0,1\}^{2n}$$

takes as input an initial curve E_0, a hash m, a signature $\sigma = (\sigma_1, ..., \sigma_l)$ and a random value x, and outputs the associated Winternitz public key p. The details is as follows:

- Exploit the function $checksumed(m)$ to split the hash m into l blocks $b_1||b_2||...||b_l$.
- Compute the public value $p_i \leftarrow H^{w-b_i-1}(E_0, \sigma_i||x)$ for $i \in \{1, ..., l\}$.
- Compute $p \leftarrow L - tree(E_0, p_1, ..., p_l)$.

D. THE SIGNATURE BASED ON SUPERSINGULAR ELLIPTIC CURVE ISOGENIES

We adopt the optimal Gravity-SPHINCS scheme proposed by Aumasson *et al* [19], and replace the hash functions, keyed one-way function, batch signature, PORST signature, WOTS+ instances with those from supersingular elliptic curve isogenies and make some changes.

Key Generation: To generate keys, we first choose a random value $x \xleftarrow{\$} \{0,1\}^n$ for the computation of keyed one-way functions, and select a secret value *seed* to deduce the secret keys of WOTS+ and PORST signature. To generate the public key, we compute a Merkle tree of high $c+h$ and take the root node as part of public key. The process is as follows:

- For $i \in \{1, ..., 2^{c+h}\}$, generate Winternitz public keys

$$\mathbf{A}_i = make - addr(0, i),$$
$$x_i \leftarrow WOTS + genpk(E_0, seed, \mathbf{A}_i, x).$$

- Compute the root of a Merkle tree with 2^{c+h} leaves and take it as part of a public key,

$$E_0' \leftarrow Merkle - root_{c+h}(E_0, (x_0, ..., x_{2^{c+h}-1})).$$

The private key is $seed \in \{0,1\}^n$ and the public key is

$$(E_0, E_0', x) \in \{0,1\}^{2n} \times \{0,1\}^{2n} \times \{0,1\}^n.$$

Signing: Given a message $M_e \in \{M_1, ..., M_i\}$, we need four steps to sign it. They are batch signing, PORST signing, WOTS+ instances and key caching.

For a sequence of messages $(M_1, ..., M_i)$ with $0 < i \leq 2^b$, the batch signing is as follows.

- For $j \in \{1, ..., i\}$, compute the message hash

$$m_j \leftarrow H(E_0, M_j),$$

- For $j \in \{i+1, ..., 2^b\}$, set $m_j \leftarrow m_1$,
- Compute a root $m \leftarrow Merkle - root(E_0, m_1, ..., m_{2^b})$,
- For the message M_e, find the authentication path

$$A_{batch} \leftarrow Merkle - auth_b(m_1, ..., m_{2^b}, e).$$

Sign the root m and take A_{batch} and e as a part of our signature.

At the dth layer, we generate a Merkle tree of height $\lceil \log_2 t \rceil$ and target at k leaves chosen from t leaves for the PORST signing.

- Compute a hyper-tree index $\lambda \in N$ and k distinct indices x_i

$$\lambda, (x_1, ..., x_k) \leftarrow PORS(E_0, m).$$

- For $i = 1$ to t, set $a = make - addr(d, 0)$,
  - compute $A_i = incr - addr(a, i)$ and pad A_i to n bits,
  - compute $s_i = G(seed, A_i)$.

  For $j = 1$ to k,
  - set the signature value $\sigma_j = s_{x_j}$,
  - compute the authentication octopus and root as $(oct, p) \leftarrow Octopus - auth_{log_2 t}(E_0, s_1, ..., s_t, x_1, ..., x_k)$.

  Then the signature is $\sigma_{d,w} = (\sigma_1, ..., \sigma_k, oct, p)$.

For $i \in \{d-1, ..., 0\}$, we do the WOTS+ instances as follows.

- Set the address $\mathbf{A}_i = make - address(i, \lambda)$.
- Sign the message p.
  $\sigma_{i,w} \leftarrow WOTS + (E_0, seed, \mathbf{A}_i, p, x)$,
  $p \leftarrow WOTS + extractpk(E_0, \mathbf{A}_i, \sigma_{i,w}, x)$,
- $\lambda' \leftarrow \lfloor \lambda / 2^h \rfloor$.
- For $u \in \{0, ..., 2^h - 1\}$, compute the WOTS+ public key.
  - let address $\mathbf{A}_{iu} = make - addr(i, (2^h\lambda' + u))$,
  - compute $p_u \leftarrow WOTS + genpk(E_0, seed, \mathbf{A}_{iu}, x)$.
- Compute the Merkle authentication

$$A_i \leftarrow Merkle - auth_h(p_0, ..., p_{2^h-1}, \lambda - 2^h\lambda').$$

- Set $\lambda \leftarrow \lambda'$.

The key caching is performed as follows.

- For $0 \leq u < 2^{c+h}$, compute the WOTS+ public key,
  - let the address be $\mathbf{A}_u = make - addr(0, u)$,
  - compute $p_u \leftarrow WOTS + genpk(E_0, seed, \mathbf{A}_u, x)$.
- Compute the Merkle authentication

$$(a_1, ..., a_{h+c}) \leftarrow Merkle - auth_{h+c}(E_0, p_0, ..., p_{2^{c+h}-1}, 2^h\lambda)$$

- Set $A_{cache} \leftarrow (a_{h+1}, ..., a_{h+c})$.

The signature is

$$(e, A_{batch}, \sigma_{d,w}, oct, \sigma_{d-1,w}, A_{d-1}, ..., \sigma_{0,w}, A_0, A_{cache}).$$

Verification: V takes as input a hash M_e, public keys E'_0 and a signature

$$(e, A_{batch}, \sigma_{d,w}, oct, \sigma_{d-1,w}, A_{d-1}, ..., \sigma_{0,w}, A_0, A_{cache})$$

and verifies it as follows:

- Compute the root of batched messages

$$m \leftarrow Merkle - extract_b(M_e, E_0, e, A_{batch}).$$

- Compute the PORST public key p,
  - compute the hyper-tree index and random subset,

$$\lambda, (x_1, ..., x_k) \leftarrow PORS(m, E_0),$$

  - compute the PORST public key, given the oct, $\sigma_{d,w}$, $(x_1, ..., x_k)$ and E_0, compute

$$p \leftarrow Octopus - auth_{log_2 t,k}(E_0, oct, \sigma_{d,w}, (x_1, ..., x_k)).$$

- If $p = \perp$, then abort and return 0.
- Verify the WOTS+ instances, for $i \in \{d-1, ..., 0\}$ do the following,
  - compute the WOTS+ public key

$$p \leftarrow WOTS + extractpk(E_0, p, \sigma_{i,w}, x),$$

  - set $\lambda' \leftarrow \lfloor \lambda / 2^h \rfloor$,
  - compute the Merkle root

$$p \leftarrow Merkle - extract_h(E_0, p, \lambda - 2^h\lambda', A_i),$$

  - set $\lambda \leftarrow \lambda'$.
- Compute the Merkle root

$$p \leftarrow Merkle - extract_c(E_0, p, \lambda, A_{cache}).$$

The result is 1 if $p = E'_0$, and 0 otherwise.

E. PARAMETER SIZES

In order to achieve λ bits of post-quantum security (see. Sec.2.1), we can set $\lceil \log_2 p \rceil \approx 4\lambda$ over $\mathbb{F}_{p^2}$.

Note that all supersingular curves are defined over $\mathbb{F}_{p^2}$ and represented in Montgomery form $By^2 = x^3 + Ax^2 + x$ where the A-coefficient suffices for the computations of isogenous curves. Meanwhile, a point on the curve can be represented only by its x-coordinate to complete the computations of scalar multiplications and isogenies. Since each field element requires 8λ bits over $\mathbb{F}_{p^2}$, in both cases, we only need one field element.

Public Keys: The public key includes the initial curve E_0, the root node E'_0 and the random x, which require 20λ bits of storage.

Private Keys: The private key can be stored as an element $seed \in \{0, 1\}^{4\lambda}$, requiring 4λ bits.

Signatures: Our signature has the form

$$(e, A_{batch}, \sigma_{d,w}, oct, \sigma_{d-1,w}, A_{d-1}, ..., \sigma_{0,w}, A_0, A_{cache}).$$

According to the signature scheme, these authentications A_{batch}, $A_i (0 \leq i \leq d-1)$ and A_{cache} separately include b curves, h curves and c curves. The Octopus authentication oct at most includes $k(\log_2 t - \lfloor \log_2 k \rfloor)$ curves. $\sigma_{d,w}$ and

$\sigma_{i,w}(0 \le i \le d-1)$ include k curves and l curves, respectively. The index e is a b-bit number. Thus the entire signature has size roughly

$$b + (b + k + k(\log_2 t - \lfloor \log_2 k \rfloor) + (l + h)d + c)8\lambda$$

bits at most.

For instance, to achieve 128 bits of post-quantum security over $\mathbb{F}_{p^2}$, the size of p is at least 521 bits, the PORST set size t is at least 16 bits and subset size $k = 28$. Other parameters can be set flexibly according to the number of signatures. In order to achieve the capacity of signing 2^{64} messages per key pair, we can set $b = 0$, $h = 14$, $d = 4$, $c = 8$, $w = 2^4$, then the signature requires $12\lambda = 962808$ bits at most.

Since we have shown the security of WOTS+, the security of the whole scheme can be proved in the same way as the method in [19], we will not go into the details here.

V. IMPLICATIONS RESULTS AND EFFICIENCY ANALYSIS

We used Visual Studio 2015 in Windows 10 on a computer with 2.90GH Intel Core i7-7700T CPU and 8G RAM to implement[1] our scheme. The details were partly referred to the implementation process of the signature scheme based on hash function proposed by Aumasson *et al.* [19] and OpenSSL 1.0.0e for the implementation of prime field $\mathbb{F}_p$ where $p = 2^{521} - 1$. The initial curve E_0 is chosen by starting a random walk from a special curve $y^2 = x^3 + 6x^2 + x$.

TABLE 3. Concrete efficiency comparison with other signatures at least 128-bit security levels. The notations Sig, Sk and Pk denote the signature, secret key and public key, respectively. All sizes are in bits.

	Sig size	Sk size	Pk size	security levels
DFJP +FS [8]	196608	256	3584	Classical
Sec4+FS [8]	180224	256	768	Classical
DFJP +U [8]	1179648	384	5376	Quantum
Sec4+U [8]	1228800	512	1536	Quantum
Yoo [4]	983040	384	2688	Quantum
SeaSign [10]	7552	256	33030144	Classical
CSI-Fish [13]	2104	128	33554432	Classical
This work	757534	521	2606	Quantum

Next, we compare our scheme, which can sign 2^{50} messages per key pair, with the signature schemes based on supersingular elliptic curve in TABLE 3. In the light of the introduction mentioned before, DFJP+U, Sec4+U as well as the Yoo's schemes [4] attain 128-bit quantum security levels, while DFJP+FS, Sec4+ FS, SeaSign and CSI-Fish schemes attain 128-bit classical security levels. The results in TABLE 3 show that the signature size of our scheme is shorter than those post-quantum signature schemes based on supersingular elliptic curve isogenies. Thus, when there are fewer messages required signatures, our scheme has more obvious advantages.

[1]See https://github.com/sysu-sidh/An-Efficient-Signature-Scheme-from-Supersingular-Elliptic-Curve-Isogenies

VI. CONCLUSIONS

We propose a more efficient hash function from supersingular elliptic curve isogeny. Based on the hash function, a new signature scheme is designed. In contrast to other post-quantum signature schemes based on supersingular elliptic curve isogeny, the scheme has smaller signature size. Furthermore, all our operations are over finite fields $\mathbb{F}_{p^2}$ where $p = 2^{521} - 1$. It is well known that the addition, multiplication, squaring and inversion operations on it are fast.

The major limitation lies in the fact that the efficient representation of supersingular elliptic curve, i.e., the output length of our hash function, significantly affects the signature size. In our setting, each supersingular elliptic curve is represented in terms of $j-$invariants and needs $2n$ bits over $\mathbb{F}_{p^2}$ with $\lceil \log_2 p \rceil = n$. Theoretically, there are about $p/12$ supersingular $j-$invariants [24], each one can be represented by n bits. Future work should be done to investigate the efficient representation of the elliptic curve $j-$invariants over $\mathbb{F}_{p^2}$.

APPENDIX

A. TRANSFORMATION BETWEEN E AND K^{SQR}

1) CONVERT THE MONTGOMERY CURVE E OVER $\mathbb{F}_{p^2}$ INTO KUMMER SURFACES K^{Sqr} OVER $\mathbb{F}_p$

Converting the Montgomery curve $E_\alpha : y^2 = x(x - \alpha)(x - 1/\alpha)$ to squared Kummer surface needs three steps.

The first step is to convert the Montgomery curve E_α into Jacobian J_{C_α}. According to Proposition 1 [31], the Weil restriction of scalars of $E_\alpha(\mathbb{F}_{p^2})$ with respect to $\mathbb{F}_{p^2}/\mathbb{F}_p$ is (2, 2)- isogenies to the Jacobian J_{C_α} of

$$C_\alpha/\mathbb{F}_p : y^2 = (x-z_1)(x-z_2)(x-z_3)(x-z_4)(x-z_5)(x-z_6)$$

where

$$\gamma^2 = \alpha;\ \beta^2 = \frac{(\alpha^2 - 1)}{\alpha}.$$

Write $\beta = \beta_0 + \beta_1 i$ and $\gamma = \gamma_0 + \gamma_1 i$ with

$$z_1 := \frac{\beta_0}{\beta_1};\ z_2 := \frac{\gamma_0}{\gamma_1};\ z_3 := -\frac{\gamma_0}{\gamma_1};\ z_4 := -\frac{\beta_1}{\beta_0};\ z_5 := -\frac{\gamma_1}{\gamma_0};$$
$$z_3 := \frac{\gamma_1}{\gamma_0}.$$

Converting a Montgomery curve $y^2 = x^3 + Ax^2 + x$ into the form $y^2 = x(x - \alpha)(x - 1/\alpha)$ where $\alpha = \frac{-A+\sqrt{A^2-4}}{2}$ and $1/\alpha = \frac{-A-\sqrt{A^2-4}}{2}$ costs one square root, one squaring, one inversion and two multiplications over $\mathbb{F}_{p^2}$. Computing γ and β cost two square roots, one inversion, one multiplication and one squaring and z_1, z_2, z_3, z_4,z_5, z_6 cost 6 inversions and 6 multiplications. Thus this step in total costs 3 square roots, 7 inversions, 9 multiplications and two squarings, i.e., $4138\,M + 5S$.

The second step is to convert the curve C_α into the Rosenhain form $C_{\lambda,\mu,\nu} : y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu)$ by an isomorphism. By setting $a = z_2 - z_4$, $b = -az_1$, $c = z_2 - z_1$

and $d = -cz_4$, the isomorphism is

$$\kappa_{a,b,c,d} : C_\alpha \to C_{\lambda,\mu,\nu}$$
$$(x, y) \to ((\frac{\beta_0\gamma_0 + \beta_1\gamma_1}{\gamma_0\beta_1 - \gamma_1\beta_0})(\frac{\beta_1 x - \beta_0}{\beta_0 x - \beta_1}), ey(\frac{\beta_0\beta_1\gamma_1}{(\beta_1\gamma_0 - \beta_0\gamma_1)(\beta_0 x + \beta_1)})^3)$$

with $e^2 = ac(a-c)(a-\nu c)(a-\mu c)(a-\lambda c)$, and

$$\lambda := \frac{(\beta_0\gamma_1 + \beta_1\gamma_0)(\beta_0\gamma_0 + \beta_1\gamma_1)}{(\gamma_0\beta_0 - \gamma_1\beta_1)(\gamma_0\beta_1 - \gamma_1\beta_0)};$$
$$\mu := \frac{(\beta_0\gamma_0+\beta_1\gamma_1)(\beta_0\gamma_0-\beta_1\gamma_1)}{(\gamma_0\beta_1+\gamma_1\beta_0)(\gamma_0\beta_1-\gamma_1\beta_0)}; \quad \nu := \frac{(\beta_0\gamma_0 + \beta_1\gamma_1)^2}{(\gamma_0\beta_1 - \gamma_1\beta_0)^2}.$$

This step only needs to compute λ, μ and ν, which costs $308\,M + 2S$.

The third step is to convert the Rosenhain form into the squared Kummer surface K^{Sqr}, i.e.,

$$K^{Sqr} : F \times X_1X_2X_3X_4 = (X_1^2 + X_2^2 + X_3^2 + X_4^2 - G(X_1 + X_2)(X_3 + X_4) - H(X_1X_2 + X_3X_4))^2$$

where

$$\mu_1 = (\frac{\gamma_0^2 - \gamma_1^2}{\gamma_0^2 + \gamma_1^2}) \cdot \sqrt{\lambda}; \mu_2 = (\frac{\gamma_0^2 - \gamma_1^2}{\gamma_0^2 + \gamma_1^2})/\sqrt{\lambda}; \mu_3 = \mu_4 = 1$$

and

$$F := 4\mu_1\mu_2\frac{(\mu_1 + \mu_2 + 2)^2(\mu_1 + \mu_2 - 2)}{(\mu_1\mu_2 - 1)^2}; G := \mu_1 + \mu_2;$$
$$H := \frac{\mu_1^2 + \mu_2^2 - 2}{\mu_1\mu_2 - 1}.$$

This step needs to compute μ_1, μ_2, μ_3, μ_4 and F, G, H, which costs $1450M + 7S$.

Henceforth, converting a Montgomery curve E over $\mathbb{F}_{p^2}$ to Kummer surfaces K^{Sqr} Over $\mathbb{F}_p$ costs $5896M + 14S$.

2) MAPPING THE POINT R ON E INTO THE POINT $R_{K^{SQR}}$ on K^{Sqr}

Mapping a point R on E into the point $R_{K^{Sqr}}$ on K^{Sqr} needs two steps: The first step is to map the point R on E into the divisor $R_{J_{C_\alpha}}$ on J_{C_α}. This can be performed by the mapping [31]

$$\eta : E \to J_{C_\alpha}$$
$$R \to (\tau \circ \rho \circ \psi(R)),$$

where ψ, ρ and τ are defined as follows:

$$\psi : E \to \widehat{E}$$
$$(x, y) \to ((\widehat{\beta}/\beta)^2 x + r_1, (\widehat{\beta}/\beta)^3 y),$$

with $\widehat{E} : y^2 = (x-r_1)(x-r_2)(x-r_3), r_1 = (\alpha - 1/\alpha)^{p-1}, r_2 = \alpha^{p-1}, r_3 = 1/\alpha^{p-1}$,and $\widehat{\beta}^2 = r_3 - r_2$. Computing ψ costs $4M + 1563S + I$.

$$\rho : \widehat{E} \to J_{C_\alpha}(\mathbb{F}_{p^2})$$
$$(\widehat{x}, \widehat{y}) \to (x^2 + u_1x + u_0, v_1x + v_0),$$

with $u_1 = 2i(\frac{\widehat{x}+1}{\widehat{x}-1})$, $u_0 = -1$, $v_1 = -4i\frac{\widehat{y}(\widehat{x}+3)}{w(\widehat{x}-1)^2}$ and $v_0 = \frac{4\widehat{y}}{w(\widehat{x}-1)}$. Computing ρ costs $6M + S + I$.

$$\tau : J_{C_\alpha}(\mathbb{F}_{p^2}) \to J_{C_\alpha}(\mathbb{F}_p)$$
$$(x^2 + u_1x + u_0, v_1x + v_0) \to (x^2 + u_1x + u_0, v_1x + v_0) \oplus (x^2 + u_1^p x + u_0^p, v_1^p x + v_0^p)$$

where $\oplus$ denotes the addition law in $J_{C_\alpha}(\mathbb{F}_{p^2})$. Computing τ costs $29M + 2091S$.

The second step is to use Algorithm 3 in [32] to translate the point on $J_{C_\alpha}(\mathbb{F}_p)$ into $K^{Sqr}(\mathbb{F}_p)$, which costs $12m + 1s + 11a$. Thus mapping the point R on E into the point $R_{K^{Sqr}}$ on K^{Sqr} needs $243M + 3655S$.

3) MAPPING THE POINT $R_{K^{SQR}}$ ON K^{Sqr} INTO THE POINT R ON E

Firstly, we map the point $R_{K^{Sqr}}$ on K^{Sqr} into $R_{J_{C_\alpha}}$ on $J_{C_\alpha}(\mathbb{F}_p)$ by Algorithm 8 in [32], which costs $243M + 12S$. Then we use the mapping $\widehat{\eta}$ to map the point $R_{J_{C_\alpha}}$ on J_{C_α} into the point R on E.

$$\widehat{\eta} : J_{C_\alpha} \to E_\alpha(\mathbb{F}_{p^2}))$$
$$R_{J_{C_\alpha}} \to \psi^{-1} \circ \oplus_E \circ \widehat{\rho}(P)$$

where $\widehat{\rho}$ and ψ^{-1} are defined as follows:

$$\widehat{\rho} : J_{C_\alpha}(\mathbb{F}_p)) \to \widehat{E}_\alpha(\mathbb{F}_{p^2}) \times \widehat{E}_\alpha(\mathbb{F}_{p^2})$$
$$P \to ((\omega \circ \phi^{-1})(x_1, y_1), (\omega \circ \phi^{-1})(x_2, y_2)),$$

with

$$\phi^{-1} : C_\alpha(\mathbb{F}_{p^2}) \to \widehat{C}_\alpha(\mathbb{F}_{p^2})$$
$$(x, y) \to (-\frac{x - i}{x + i}, -i\frac{yw}{(x + i)^3})$$

and

$$\omega : \widehat{C_\alpha} \to \widehat{E_\alpha}$$
$$(x, y) \to (x^2, y);$$
$$\psi^{-1} : \widehat{E}_\alpha \to E_\alpha$$
$$(x, y) \to (\beta/\widehat{\beta})^2(x - r_1), (\beta/\widehat{\beta})^3 y).$$

The map $\widehat{\eta}$ needs $478M + 4S$. Thus mapping the point $R_{K^{Sqr}}$ on K^{Sqr} into the point R on E costs $721M + 16S$.

4) OPERATIONS ON KUMMER SURFACES

When converting the Montgomery curves into Kummer surfaces, the scalar multiplications and isogeny computations are all calculated on Kummer surfaces and are comprised of three sub-operations. Define $H : \mathbb{P}^3 \to \mathbb{P}^3$ as the 4-way Hadamard transform in $\mathbb{P}^3$, i.e.,

$$H : (l_1 : l_2 : l_3 : l_4) \to (l_1 + l_2 + l_3 + l_4 : l_1 + l_2 - l_3 - l_4 : l_1 - l_2 + l_3 - l_4 : l_1 - l_2 - l_3 + l_4)$$

together with the coordinate squaring operation $S : \mathbb{P}^3 \to \mathbb{P}^3$, as

$$S : (l_1 : l_2 : l_3 : l_4) \to ((l_1^2 : l_2^2 : l_3^2 : l_4^2))$$

and the coordinate scaling operation $C_{d_1:d_2:d_3:d_4} : \mathbb{P}^3 \to \mathbb{P}^3$, as

$$C_{d_1:d_2:d_3:d_4} : (l_1 : l_2 : l_3 : l_4) \to (l_1/d_1 : l_2/d_2 : l_3/d_3 : l_4/d_4) = (\pi_1 l_1 : \pi_2 l_2 : \pi_3 l_3 : \pi_4 l_4)$$

where $\pi_i = d_1 d_2 d_3 d_4 / d_i$ for $i \in \{1, 2, 3, 4\}$. It follows that performing the Hadamard transform needs at most 8 field additions, the coordinate squaring operation needs at most 4 field squarings and the coordinate scaling operation needs at most 10 field multiplications.

Computing (2, 2)-isogenies between Kummer surfaces were derived in [31]. Like the general case as in [31], the 2-isogeny corresponds to 2-torsion point which is not (0, 0) on Montgomery curve, the corresponding (2,2)-isogeny can be computed by finding a point $Q \in K^{Sqr}$ of order 4 such that $P = 2Q \in \{\Upsilon, \widehat{\Upsilon}\}$ where Υ or $\widehat{\Upsilon}$ is the (2,2)-kernel. Writing $Q' = H(Q) = \{Q'_1 : Q'_2 : Q'_3 : Q'_4\}$ and $P' = H(P) = \{P'_1 : P'_2 : P'_3 : P'_4\}$, then

$$C_{Q,P} : (X_1 : X_2 : X_3 : X_4) \to (\pi_1 X_1 : \pi_2 X_2 : \pi_3 X_3 : \pi_4 X_4)$$

where

$$\pi_1 = P'_2 Q'_4, \quad \pi_2 = P'_1 Q'_4, \quad \pi_3 = \pi_4 = P'_2 Q'_1$$

when $P \in \{\Upsilon_1, \widehat{\Upsilon}_1\}$ or

$$\pi_1 = P'_2 Q'_3, \quad \pi_2 = P'_1 Q'_3, \quad \pi_3 = \pi_4 = P'_2 Q'_1$$

when $P \in \{\Upsilon_2, \widehat{\Upsilon}_2\}$.

Then the (2,2)-isogeny can be derived from [31]. Let Q be a point of order 4, $P = 2Q$ and $P \in \{\Upsilon, \widehat{\Upsilon}\}$. Denote by

$$\varphi_P : K^{Sqr} \to K^{Sqr}/\{\Upsilon, \widehat{\Upsilon}\}$$
$$R \to (S \cdot H \cdot C_{Q,P} \cdot H(R))$$

the full (2,2)-isogeny.

Evaluating an 2-isogeny needs $4m + 4s + 16a$ and computing an 2-isogeny curve costs $19m + 4s + 16a$.

B. INTERNAL FUNCTIONS

Operation on Address: The function $make - addr : \{0, ..., d\} \times \mathbb{N} \to \mathrm{A}$ takes as input a layer $i \in \{0, ..., d\}$ and an index $j \in \mathbb{N}$ and returns $a = (i, j \mod 2^{c+dh}, 0) \in \mathrm{A}$.

The function $incr - addr : \mathrm{A} \times \mathbb{N} \to \mathrm{A}$ takes as input an address $a = (i, j, \lambda)$ and an integer x and returns the address $a' = (i, j, \lambda + x) \in \mathrm{A}$.

L-Tree: The function L-tree: $\{0,1\}^{2n} \times \{0,1\}^{2nl} \to \{0,1\}^{2n}$ takes as input a supersingular curve E_0 and l leaves hashes $x_i \in \{0,1\}^{2n}$, and returns the associated L-tree root $r \in \{0,1\}^{2n}$, defined by recurrence as follows.

$$L - tree(x_1) = x_1.$$
$$L - tree(x_1, ..., x_{2i+2}) = L - tree(H(E_0, x_1||x_2), ..., H(E_0, x_{2i+1}||x_{2i+2})).$$
$$L - tree(x_1, ..., x_{2i+3}) = L - tree(H(E_0, x_1||x_2), ..., H(E_0, x_{2i+1}||x_{2i+2}), x_{2i+3}).$$

Merkle-Root: The function Merkle-root: $\{0,1\}^{2n} \times \{0,1\}^{2n \times 2^h} \to \{0,1\}^{2n}$ takes as input a supersingular curve E_0 and 2^h leaf hashes x_j, and outputs the associated Merkle tree root $r \in \{0,1\}^{2n}$. The process is as follows:

$$Merkle - root_0(x_0) = x_0.$$
$$Merkle - root_i(x_0, ..., x_{2^i}) = Merkle - root_{i-1}(H(E_0, x_0||x_1), ..., H(E_0, x_{2^i-1}||x_{2^i})), \quad where\ 0 < i \le h$$

Merkle Tree Authentication: The function $Merkle - auth_h : \{0,1\}^{2n} \times \{0,1\}^{2n \cdot 2^h} \times \{0, ..., 2^h - 1\} \to \{0,1\}^{2nh}$ takes as input a supersingular curve E_0, 2^h leaf hashes x_i and a leaf index $0 \le j < 2^h$, and outputs the associated Merkle tree authentication path $(a_1, ..., a_h) \in \{0,1\}^{2nh}$.

$Merkle - auth_1(x_0, x_1, j) = a_1 \leftarrow x_{j \oplus 1}$ where $\oplus$ denotes the bitwise XOR operation on non-negative integers.

$Merkle - auth_h(x_0, x_1 ..., x_{2^h}, j)$ is

$$a_1 \leftarrow x_{j \oplus 1},$$
$$a_2, ..., a_h \leftarrow Merkle - auth_{h-1}(H(E_0, x_0||x_1), ..., H(E_0, x_{2^i-1}||x_{2^i}), \lfloor j/2 \rfloor).$$

Merkle Tree Root Extraction: The function $Merkle - extract_h : \{0,1\}^{2n} \times \{0,1\}^{2n} \times \{0, ..., 2^h - 1\} \times \{0,1\}^{2nh} \to \{0,1\}^{2n}$ takes as input a supersingular curve E_0, a leaf hash x_i, a leaf index $0 \le j < 2^h$ and an authentication path $(a_1, ..., a_h)$, and outputs the associated Merkle tree root $r \in \{0,1\}^{2n}$.

$$Merkle - extract_0(x, j) = x,$$
$$Merkle - extract_h(x, j, a_1, ..., a_h) = Merkle - extrac_{h-1}(x', \lfloor j/2 \rfloor, a_2, ..., a_h)$$

where

$$x' = \begin{cases} H(E_0, x||a_1) & \text{if } j \bmod 2 = 0 \\ H(E_0, a_1||x) & \text{if } j \bmod 2 = 1. \end{cases}$$

Octopus Authentication: The function $Octopus - auth_h : \{0,1\}^{2n} \times \{0,1\}^{2n \times 2^h} \times \{0, ..., 2^h - 1\}^k \to \{0,1\}^* \times \{0,1\}^{2n}$ takes as input a supersingular curve E_0, 2^h leaf hashes x_i and k distinct octopus authentication nodes, and outputs the authentication path oct and the octopus root $r \in \{0,1\}^{2n}$. It is defined by recurrence on h as:

$Octopus - auth_0(x_0, j_1) = (\varnothing, x_0)$,

$Octopus - auth_h(x_0, x_1, ..., x_{2^h}, j_1, ..., j_k)$ is computed as

$$\begin{cases} j'_1, ..., j'_{k'} \leftarrow unique(\lfloor j_1/2 \rfloor, ..., \lfloor j_k/2 \rfloor); \\ oct', r \leftarrow Octopus - auth_{h-1}(H(E_0, x_0||x_1), ..., \\ H(E_0, x_{2^h-1}||x_{2^h}), j'_1, ..., j'_{k'}); \\ z_1, ..., z_{2k'-k} \leftarrow (j_1 \oplus 1, ..., j_k \oplus 1)/(j_1, ..., j_k); \\ a_1, ..., a_{2k'-k} \leftarrow (x_{z_1}, ..., x_{2k'-k}); \\ oct \leftarrow (a_1, ..., a_{2k'-k}). \end{cases}$$

where unique() removes duplicates in a sequence, and A/B denotes the set difference.

Octopus Root Extraction: The function $Octopus - extract_{h,k}$:

$$\{0,1\}^{2n} \times \{0,1\}^{2nk} \times \{0,1\}^* \to \{0,1\}^{2n} \bigcup \{\perp\}$$

takes as input a supersingular curve E_0, k leaf hashes $x_i \in \{0,1\}^{2n}$, k leaf indices $0 \leq j_i < 2^h$ and an authentication octopus $oct \in \{0,1\}^*$, and outputs the associated Merkle tree root $r \in \{0,1\}^{2n}$. If the number of hashes in the authentication octopus is invalid, it outputs $\perp$. It is defined by recurrence on h as:

$$Octopus - extract_{0,1}(x_1, j_1, oct) = \begin{cases} x_1 & \text{if } oct = \varnothing \\ \perp & \text{if } otherwise. \end{cases}$$

$Octopus - extract_{h,k}(x_1, ..., x_k, j_1, ..., j_k, oct)$ is computed as

$$\begin{cases} j'_1, ..., j'_{k'} \leftarrow unique(\lfloor j_1/2 \rfloor, ..., \lfloor j_k/2 \rfloor); \\ L \leftarrow Oct - layer((x_1, j_1), ..., (x_k, j_k), oct); \\ \perp \quad \text{if} \quad L = \perp; \\ Octopus - extract_{h-1,k'}(x'_1, ..., x'_{k'}, j'_1, ..., j_{k'}, oct'); \\ \text{if} \quad L = (x_1, ..., x_{k'}, oct'). \end{cases}$$

where Oct-layer() is defined by recurrence as:

$$oct - layer(x_1, j_1, oct) = \begin{cases} \perp & \text{if } oct = \varnothing \\ H(E_0, x_1||a), oct' & \text{if } oct = (a, oct) \bigwedge j_1 \bmod 2 = 0 \\ H(E_0, a||x_1), oct' & \text{if } oct = (a, oct) \bigwedge j_1 \bmod 2 = 1. \end{cases}$$

$Oct - layer(x_1, j_1, x_2, j_2, ..., x_k, j_k, oct)$ is

$$\begin{cases} H(E_0, x_1||x_2), oct - layer(x_3, j_3, ..., x_k, j_k, oct) \\ \qquad \text{if } j_1 \oplus 1 = j_2; \\ \perp \qquad \text{if } j_1 \oplus 1 \neq j_2 \wedge oct = \varnothing; \\ H(E_0, x_1||a), oct - layer(x_2, j_2, ..., x_k, j_k, oct') \\ \qquad \text{if } oct = (a, oct) \wedge j_1 \mod 2 = 0; \\ H(E_0, a||x_1), oct - layer(x_2, j_2, ..., x_k, j_k, oct') \\ \qquad \text{if } oct = (a, oct) \wedge j_1 \mod 2 = 1. \end{cases}$$

PORS: This function PORS: $\{0,1\}^{2n} \times \{0,1\}^{2n} \to N \times T^k$ takes as input a supersingular curve E_0 and a hash m, and outputs a hyper-tree index $\lambda \in N$ and k distinct indices x_i as follows:

- Compute $s = H(E_0, m) \mod 2^n$.
- Set address $A = make - addr(d, 0)$ and pad A to n bits.
- Compute an index $\lambda = G(s, A) \mod 2^{c+dh}$.
- Initialize $X \leftarrow \emptyset$ and $j = 0$.
- While $|X| < k$ do the following:
  - increment $j \leftarrow j + 1$,
  - compute address $A_j = incr - addr(A, j)$ and pad A_j to n bits,
  - compute $u = G(s, A_j)$ and split u into $v = \lfloor n/\log_2 t \rfloor$ blocks, namely $u = u_1||u_2||...u_v$,
  - for $i \in \{1, ..., v\}$, if $|X| < k$ update $X \leftarrow X \bigcup u_i$,
- Compute $(x_1, ..., x_k) \leftarrow sorted(X)$.

REFERENCES

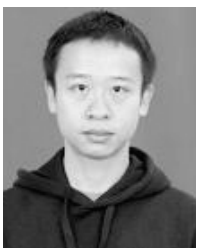
[1] S. D. Galbraith and F. Vercauteren, "Computational problems in supersingular elliptic curve isogenies," *Quantum Inf. Process*, vol. 17, no. 10, p. 265, Oct. 2018.
[2] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *Proc. PQCrypto*, Taibei, Taiwan, 2011, pp. 19–34.
[3] D. Jao *et al. Supersingular Isogeny Key Encapsulation*. Accessed: 2019. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions/SIKE.zip
[4] Y. Yoo, R. Azarderakhsh, A. Jalali, D. Jao, and V. Soukharev, "A post-quantum digital signature scheme based on supersingular isogenies," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Sliema, Malta, 2017, pp. 163–181.
[5] J. F. Biasse, D. Jao, and A. Sankar, "A quantum algorithm for computing isogenies between supersingular elliptic curves," in *Proc. INDOCRYPT*, New Delhi, India, 2014, pp. 428–442.
[6] D. Jao and V. Soukharev, "Isogeny-based quantum-resistant undeniable signatures," in *Proc. PQCrypto*, Waterloo, ON, Canada, 2014, pp. 19–34.
[7] X. Sun, H. Tian, and Y. Wang, "Toward quantum-resistant strong designated verifier signature," *Int. J. Grid Utility Comput.*, vol. 5, no. 2, pp. 80–86, Mar. 2014.
[8] S. D. Galbraith, C. Petit, and J. Silva, "Identification protocols and signature schemes based on supersingular isogeny problems," in *Proc. ASIACRYPT*, Hong Kong, 2017, pp. 3–33.
[9] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, "CSIDH: An efficient post-quantum commutative group action," in *Proc. ASIACRYPT*, Brisbane, QLD, Australia, 2018, pp. 395–427.
[10] L. De Feo and S. D. Galbraith, "SeaSign: Compact isogeny signatures from class group actions," in *Proc. EUROCRY*, Darmstadt, Germany, 2019, pp. 759–789. [Online]. Available: https://eprint.iacr.org/2018/824
[11] T. Deru, L. Panny, and F. Vercauteren, "Faster seasign signatures through improved rejection sampling," in *Proc. PQCrypto*, Chongqing, China, 2019, pp. 271–285. [Online]. Available: https://eprint.iacr.org/2018/1109
[12] T. Kleinjung, "Quadratic sieving," *Math. Comp.*, vol. 85, no. 300, pp. 1861–1873, 2016.
[13] W. Beullens, T. Kleinjung, and F. Vercauteren, "CSI-FiSh: Efficient isogeny based signatures through class group computations," Cryptol. ePrint Arch., Tech. Rep. 2019/498.
[14] X. Bonnetain and A. Schrottenloher, "Submerging CSIDH cryptology," ePrint Arch., Tech. Rep. 2018/1059, 2018. [Online]. Available: https://eprint.iacr.org/2018/537
[15] C. Peikert, "He gives C-sieves on the CSIDH," Cryptol. ePrint Arch., Tech. Rep. 2019/725, 2018. [Online]. Available: https://eprint.iacr.org/2019/725
[16] D. Unruh, "Post-quantum security of Fiat–Shamir," in *Proc. ASIACRYPT*, Hong Kong, 2017, pp. 65–95.
[17] D. Unruh, "Non-interactive zero-knowledge proofs in the quantum random oracle model," in *Proc. EUROCRYPT*, Sofia, Bulgaria, 2015, pp. 755–784.
[18] J. D. Bernstein *et al.* (2017). *SPHINCS +*. [Online]. Available: https: //csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions/SPHINCS+.zip
[19] J. P. Aumasson and G. Endignoux. (2017). *Design and Implementation of a Post-Quantum Hash-Based Cryptographic Signature Scheme*. [Online]. Available: https: //csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions/Gravity-SPHINCS.zip
[20] D. X. Charles, K. E. Lauter, and E. Z. Goren, "Cryptographic hash functions from expander graphs," *J. Cryptol.*, vol. 22, no. 1, pp. 93–113, 2009.
[21] K. Eisentrager, S. Hallgren, K. Lauter, T. Morrison, and C. Petit, "Supersingular isogeny graphs and endomorphism rings: Reductions and solutions," in *Proc. EUROCRYPT*, Tel Aviv, Israel, 2018, pp. 329–368.
[22] J. Doliskani, G. C. C. F. Pereira, and P. S. L. M. Barreto, "Faster cryptographic hash function from supersingular isogeny graphs," Cryptol. ePrint Arch., Rep. 2017/1202, 2017. [Online]. Available: https://eprint.iacr.org/2017/1202.
[23] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing, and M. Rückert, "On the security of the winternitz one-time signature scheme," *Int. J. Appl. Cryptogr.*, vol. 3, pp. 363–378, Jul. 2011.
[24] J. H. Silverman, *The Arithmetic of Elliptic Curves*. New York, NY, USA: Springer, 2009.
[25] W. C. Waterhouse and J. S. Milne, "Abelian varieties over finite fields," Univ. Nguyen, Cambridge, MA, USA, Tech. Rep., 1968, vol. 6, no. 1.
[26] J. Vélu, "Isogénies entre courbes elliptiques," *Comptes Rendus de l'Académie des Sciences de Paris*, vol. 273, pp. 238–241, 1971.

[27] S. Hoory, N. Linial, and A. Wigderson, "Expander graphs and their applications," *Bull. Amer. Math. Soc.*, vol. 43, pp. 439–561, Aug. 2006.
[28] P. N. Chung, C. Costello, and B. Smith, "Fast, uniform scalar multiplication for genus 2 jacobians with fast kummers," in *Proc. SAC*, vol. 10532, 2017, pp. 465–481.
[29] I. B. Damgård, "A design principle for hash functions," in *Proc. CRYPTO*, Santa Barbara, CA, USA, 1989, pp. 416–427.
[30] J. Scholten. *Weil Restriction of an Elliptic Curve Over a Quadratic Extension*. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.7987& rep=rep1&type=pdf
[31] C. Costello, "Computing supersingular isogenies on kummer surfaces," in *Proc. ASIACRYPT*, Brisbane, QLD, Australia, 2018, pp. 428–456.
[32] J. Renes, P. Schwabe, B. Smith, and L. Batina, "μKummer: Efficient hyperelliptic signatures and key exchange on microcontrollers," in *Proc. CHES*, Santa Barbara, CA, USA, 2016, pp. 301–320.
[33] L. De Feo, D. Jao, and J. Plût, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," *J. Math. Cryptol.*, vol. 8, no. 3, pp. 209–247, 2014.
[34] S. Kölbl, M. M. Lauridsen, F. Mendel, and C. Rechberger. *Haraka v2-Efficient Short-Input Hashing for Post-Quantum Applications*. [Online]. Available: https://eprint.iacr.org/2016/098
[35] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. New York, NY, USA: Taylor & Francis, 2014, pp. 174–182.
[36] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.
[37] T. Malkin, D. Micciancio, and S. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods," in *Proc. EUROCRYPT*, Amsterdam, The Netherlands, 2002, pp. 400–417.
[38] H. Hisil and C. Costello, "Jacobian coordinates on genus 2 curves," in *Proc. ASIACRYPT*, vol. 8873, pp. 338–357, 2016.
[39] R. Granger and M. Scott, "Faster ECC over $\mathbb{F}_2^{521} - 1$," in *Proc. PKC*, Gaithersburg, MD, USA, 2015, pp. 539–553.

YAN HUANG was born in 1988. She received the B.S. degree from the School of Mathematics, South China Normal University, China, in 2015. She is currently pursuing the Ph.D. degree with the School of Communication Engineering, Sun Yat-sen University, Guangzhou, China. Her research interest includes isogeny-based cryptography.

FANGGUO ZHANG received the Ph.D. degree from the School of Communication Engineering, Xidian University, in 2001. From 2003 to 2004, he was a Research Fellow with the school of Information Technology and Computer Science, Wollongong University, Australia. He is currently a Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. He is also the Standing Director of the China Association for Cryptologic Research and the Co-Director of the Guangdong Key Laboratory of Information Security Technology. His research interests include cryptography and its applications, especially elliptic curve cryptosystems, secure multi-party computations, and the provable security of elliptic curve cryptosystems.

ZHIJIE LIU was born in 1993. He received the B.S. degree from the School of Computer Science, South China Normal University, China, in 2017. He is currently pursuing the M.S. degree with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interest includes elliptic curve cryptosystems.



HUANG ZHANG was born in 1988. He received the M.S. degree from the School of Data and Computer Science, Sun Yat-sen University, China, in 2014, where he is currently pursuing the Ph.D. degree. His research interests include lattice-based cryptography and zero-knowledge.