On the Cost of Computing Isogenies Between Supersingular Elliptic Curves

Gora Adj[1], Daniel Cervantes-Vázquez[2], Jesús-Javier Chi-Domínguez[2], Alfred Menezes[1], and Francisco Rodríguez-Henríquez[2(✉)]

[1] Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada
gora.adj@gmail.com, ajmeneze@uwaterloo.ca
[2] Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico
{dcervantes,jjchi}@computacion.cs.cinvestav.mx, francisco@cs.cinvestav.mx

Abstract. The security of the Jao-De Feo Supersingular Isogeny Diffie-Hellman (SIDH) key agreement scheme is based on the intractability of the Computational Supersingular Isogeny (CSSI) problem—computing $\mathbb{F}_{p^2}$-rational isogenies of degrees 2^e and 3^e between certain supersingular elliptic curves defined over $\mathbb{F}_{p^2}$. The classical meet-in-the-middle attack on CSSI has an expected running time of $O(p^{1/4})$, but also has $O(p^{1/4})$ storage requirements. In this paper, we demonstrate that the van Oorschot-Wiener golden collision finding algorithm has a lower cost (but higher running time) for solving CSSI, and thus should be used instead of the meet-in-the-middle attack to assess the security of SIDH against classical attacks. The smaller parameter p brings significantly improved performance for SIDH.

1 Introduction

The Supersingular Isogeny Diffie-Hellman (SIDH) key agreement scheme was proposed by Jao and De Feo [14] (see also [9]). It is one of 69 candidates being considered by the U.S. government's National Institute of Standards and Technology (NIST) for inclusion in a forthcoming standard for quantum-safe cryptography [13]. The security of SIDH is based on the difficulty of the Computational Supersingular Isogeny (CSSI) problem, which was first defined by Charles, Goren and Lauter [4] in their paper that introduced an isogeny-based hash function. The CSSI problem is also the basis for the security of isogeny-based signature schemes [11,30] and an undeniable signature scheme [15].

Let p be a prime, let ℓ be a small prime (e.g., $\ell \in \{2, 3\}$), and let E and E' be two supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ for which a (separable) degree-ℓ^e isogeny $\phi : E \to E'$ defined over $\mathbb{F}_{p^2}$ exists. The CSSI problem is that of constructing such an isogeny. In [9], the CSSI problem is assessed as having a complexity of $O(p^{1/4})$ and $O(p^{1/6})$ against classical and quantum attacks [26], respectively. The classical attack is a meet-in-the-middle attack (MITM) that has time complexity $O(p^{1/4})$ and space complexity $O(p^{1/4})$. We observe that the

C. Cid and M. J. Jacobson, Jr. (Eds.): SAC 2018, LNCS 11349, pp. 322–343, 2019.
https://doi.org/10.1007/978-3-030-10970-7_15

(classical) van Oorschot-Wiener golden collision finding algorithm [19,20] can be employed to construct ϕ. Whereas the time complexity of the van Oorschot-Wiener algorithm is higher than that of the meet-in-the-middle attack, its space requirements are smaller. Our cost analysis of these two CSSI attacks leads to the conclusion that, despite its higher running time, the golden collision finding CSSI attack has a lower cost than the meet-in-the-middle attack, and thus should be used to assess the security of SIDH against (known) classical attacks.

The remainder of this paper is organized as follows. The CSSI problem and relevant mathematics background are presented in Sect. 2. In Sects. 3 and 4, we report on our implementation of the meet-in-the-middle and golden collision search methods for solving CSSI. Our implementations confirm that the heuristic analysis of these CSSI attacks accurately predicts their performance in practice. Our cost models and cost comparisons are presented in Sect. 5. Finally, in Sect. 6 we make some concluding remarks.

2 Computational Supersingular Isogeny Problem

2.1 Mathematical Prerequisites

Let $p = \ell_A^{e_A}\ell_B^{e_B} - 1$ be a prime[1], where ℓ_A and ℓ_B are distinct small primes and $\ell_A^{e_A} \approx \ell_B^{e_B} \approx p^{1/2}$. Let E be a (supersingular) elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. Then $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{p+1} \oplus \mathbb{Z}_{p+1}$, whence the torsion groups $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ are contained in $E(\mathbb{F}_{p^2})$.

In the following, we write (ℓ, e) to mean either (ℓ_A, e_A) or (ℓ_B, e_B). All isogenies ϕ considered in this paper are separable, whereby $\deg\phi = \#\mathrm{Ker}(\phi)$.

Let S be an order-ℓ^e subgroup of $E[\ell^e]$. Then there exists an isogeny $\phi : E \to E'$ (with both ϕ and E' defined over $\mathbb{F}_{p^2}$) with kernel S. The isogeny ϕ is unique up to isomorphism in the sense that if $\tilde{\phi} : E \to \tilde{E}$ is another isogeny defined over $\mathbb{F}_{p^2}$ with kernel S, then there exists an $\mathbb{F}_{p^2}$-isomorphism $\psi : E' \to \tilde{E}$ with $\tilde{\phi} = \psi \circ \phi$.

Given E and S, an isogeny ϕ with kernel S and the equation of E' can be computed using Vélu's formula [27]. The running time of Vélu's formula is polynomial in $\#S$ and $\log p$. Since $\#S \approx p^{1/2}$, a direct application of Vélu's formula does not yield a polynomial-time algorithm for computing ϕ and E'. However, since $\#S$ is a power of a small prime, one can compute ϕ and E' in time that is polynomial in $\log p$ by using Vélu's formula to compute a sequence of e degree-ℓ isogenies (see Sect. 2.2).

We will denote the elliptic curve that Vélu's formula yields by E/S and the (Vélu) isogeny by $\phi_S : E \to E/S$. As noted above, ϕ_S is unique up to isomorphism. Thus, for any fixed E, there is a one-to-one correspondence between order-ℓ^e subgroups of $E[\ell^e]$ and degree-ℓ^e isogenies $\phi : E \to E'$ defined over $\mathbb{F}_{p^2}$. It follows that the number of degree-ℓ^e isogenies $\phi : E \to E'$ is $\ell^e + \ell^{e-1} = (\ell+1)\ell^{e-1}$.

[1] More generally, one can take $p = \ell_A^{e_A}\ell_B^{e_B} d \pm 1$ where d is a small cofactor.

2.2 Vélu's Formula

Vélu's formula (see [4]) can be used to compute degree-ℓ isogenies. We present Vélu's formula for $\ell = 2$ and $\ell = 3$.

Consider the elliptic curve $E/\mathbb{F}_{p^2} : Y^2 = X^3 + aX + b$, and let $P = (X_P, Y_P) \in E(\mathbb{F}_{p^2})$ be a point of order two. Let $v = 3X_P^2 + a$, $a' = a - 5v$, $b' = b - 7vX_P$, and define the elliptic curve $E'/\mathbb{F}_{p^2} : Y^2 = X^3 + a'X + b'$. Then the map

$$(X, Y) \mapsto \left(X + \frac{v}{X - X_P},\ Y - \frac{vY}{(X - X_P)^2} \right)$$

is a degree-2 isogeny from E to E' with kernel $\langle P \rangle$.

Let $P = (X_P, Y_P) \in E(\mathbb{F}_{p^2})$ be a point of order three. Let $v = 6X_P^2 + 2a$, $u = 4Y_P^2$, $a' = a - 5v$, $b' = b - 7(u + vX_P)$, and define the elliptic curve $E'/\mathbb{F}_{p^2} : Y^2 = X^3 + a'X + b'$. Then the map

$$(X, Y) \mapsto \left(X + \frac{v}{X - X_P} + \frac{u}{(X - X_P)^2},\ Y \left(1 - \frac{v}{(X - X_P)^2} - \frac{2u}{(X - X_P)^3} \right) \right)$$

is a degree-3 isogeny from E to E' with kernel $\langle P \rangle$.

Suppose now that $R \in E(\mathbb{F}_{p^2})$ has order ℓ^e where $\ell \in \{2, 3\}$ and $e \geq 1$. Then the isogeny $\phi : E \to E/\langle R \rangle$ can be efficiently computed as follows. Define $E_0 = E$ and $R_0 = R$. For $i = 0, 1, \ldots, e - 1$, let $\phi_i : E_i \to E_{i+1}$ be the degree-ℓ isogeny obtained using Vélu's formula with kernel $\langle \ell^{e-1-i} R_i \rangle$, and let $R_{i+1} = \phi_i(R_i)$. Then $\phi = \phi_{e-1} \circ \cdots \circ \phi_0$.

Remark 1 (*cost of computing an ℓ^e-isogeny*). As shown in [9], a 'balanced strategy' for computing a degree-ℓ^e isogeny requires approximately $\frac{e}{2} \log_2 e$ point multiplications by ℓ, $\frac{e}{2} \log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves. Also presented in [9] is a slightly faster 'optimal strategy' that accounts for the relative costs of a point multiplication and a degree-ℓ isogeny evaluation.

2.3 SIDH

In SIDH, the parameters ℓ_A, ℓ_B, e_A, e_B, p and E are fixed and public, as are bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ for the torsion groups $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$.

In (unauthenticated) SIDH, Alice selects $m_A, n_A \in_R [0, \ell_A^{e_A} - 1]$, not both divisible by ℓ_A, and sets $R_A = m_A P_A + n_A Q_A$ and $A = \langle R_A \rangle$; note that A is an order-$\ell_A^{e_A}$ subgroup of $E[\ell_A^{e_A}]$. Alice then computes the isogeny $\phi_A : E \to E/A$ while keeping A and ϕ_A secret. She transmits

$$E/A, \quad \phi_A(P_B), \quad \phi_A(Q_B)$$

to Bob. Similarly, Bob selects $m_B, n_B \in_R [0, \ell_B^{e_B} - 1]$, not both divisible by ℓ_B, and sets $R_B = m_B P_B + n_B Q_B$ and $B = \langle R_B \rangle$. Bob then computes the isogeny $\phi_B : E \to E/B$. He keeps B and ϕ_B secret and transmits

$$E/B, \quad \phi_B(P_A), \quad \phi_B(Q_A)$$

to Alice. Thereafter, Alice computes $\phi_B(R_A) = m_A\phi_B(P_A) + n_A\phi_B(Q_A)$ and

$$(E/B)/\langle\phi_B(R_A)\rangle,$$

whereas Bob computes $\phi_A(R_B) = m_B\phi_A(P_B) + n_B\phi_A(Q_B)$ and

$$(E/A)/\langle\phi_A(R_B)\rangle.$$

The compositions of isogenies

$$E \to E/A \to (E/A)/\langle\phi_A(R_B)\rangle$$

and

$$E \to E/B \to (E/B)/\langle\phi_B(R_A)\rangle$$

both have kernel $\langle R_A, R_B\rangle$. Hence the elliptic curves computed by Alice and Bob are isomorphic over $\mathbb{F}_{p^2}$, and their shared secret k is the j-invariant of these curves.

Remark 2 (*SIDH vs. SIKE*). SIDH is an unauthenticated key agreement protocol. The NIST submission [13] specifies a variant of SIDH that is a key encapsulation mechanism (KEM) called SIKE (Supersingular Isogeny Key Encapsulation). In SIKE, Alice's long-term public key is $(E/A, \phi_A(P_B), \phi_A(Q_B))$. Bob sends Alice an ephemeral public key $(E/B, \phi_B(P_A), \phi_B(Q_A))$ where B is derived from Alice's public key and a random string, and then computes a session key from the j-invariant of the elliptic curve $(E/A)/\langle\phi_A(R_B)\rangle$, the aforementioned random string, and the ephemeral public key. One technical difference between the original SIDH specification in [9,14] and the SIKE specification in [13] (and also the SIDH implementation in [5]) is that in the latter the secret R_A is of the form $P_A + n_AQ_A$ where n_A is selected (almost) uniformly at random from the interval $[0, \ell_A^{e_A} - 1]$ (and similarly for R_B). Thus, R_A is selected uniformly at random from a subset of size approximately ℓ^{e_A} of the set of all order-$\ell_A^{e_A}$ subgroups (which has cardinality $\ell_A^{e_A} + \ell_A^{e_A-1}$).

2.4 CSSI

The challenge faced by a passive adversary is to compute k given the public parameters, E/A, E/B, $\phi_A(P_B)$, $\phi_A(Q_B)$, $\phi_B(P_A)$ and $\phi_B(Q_A)$. A necessary condition for hardness of this problem is the intractability of the Computational Supersingular Isogeny (CSSI) problem: Given the public parameters ℓ_A, ℓ_B, e_A, e_B, p, E, P_A, Q_A, P_B, Q_B, the elliptic curve E/A, and the auxiliary points $\phi_A(P_B)$ and $\phi_A(Q_B)$, compute the Vélu isogeny $\phi_A : E \to E/A$ (or, equivalently, determine a generator of A).

An assumption one makes (e.g., see [9]) is that the auxiliary points $\phi_A(P_B)$ and $\phi_A(Q_B)$ are of no use in solving CSSI. Thus, we can simplify the statement of the CSSI problem to the following:

Problem 1 (CSSI). Given the public parameters ℓ_A, ℓ_B, e_A, e_B, p, E, P_A, Q_A, and the elliptic curve E/A, compute a degree-$\ell_A^{e_A}$ isogeny $\phi_A : E \to E/A$.

3 Meet-in-the-Middle

For the sake of simplicity, we will suppose that e is even. We denote the number of order-$\ell^{e/2}$ subgroups of $E[\ell^e]$ by $N = (\ell+1)\ell^{e/2-1} \approx p^{1/4}$.

Let $E_1 = E$ and $E_2 = E/A$. Let R denote the set of all j-invariants of elliptic curves that are isogenous to E_1; then $\#R \approx p/12$ [23]. Let R_1 denote the set of all j-invariants of elliptic curves over $\mathbb{F}_{p^2}$ that are $\ell^{e/2}$-isogenous to E_1. Since $\#R \gg N$, one expects that the number of pairs of distinct order-$\ell^{e/2}$ subgroups (A_1, A_2) of $E_1[\ell^e]$ with $j(E_1/A_1) = j(E_1/A_2)$ is very small. Thus, we shall assume for the sake of simplicity that $\#R_1 = N$. Similarly, we let R_2 denote the set of all j-invariants of elliptic curves that are $\ell^{e/2}$-isogenous to E_2, and assume that $\#R_2 = N$. Since E_1 is ℓ^e-isogenous to E_2, we know that $R_1 \cap R_2 \neq \emptyset$. Moreover, since $\#R_1 \ll \#R$ and $\#R_2 \ll \#R$, it is reasonable to assume that $\#(R_1 \cap R_2) = 1$; in other words, we can assume that there is a unique degree-ℓ^e isogeny $\phi : E_1 \to E_2$.

3.1 Basic Method

The meet-in-the-middle attack on CSSI [9], which we denote by MITM-basic, proceeds by building a (sorted) table with entries $(j(E_1/A_1), A_1)$, where A_1 ranges over all order-$\ell^{e/2}$ subgroups of $E_1[\ell^e]$. Next, for each order-$\ell^{e/2}$ subgroup A_2 of $E_2[\ell^e]$, one computes E_2/A_2 and searches for $j(E_2/A_2)$ in the table (see Fig. 1). If $j(E_2/A_2) = j(E_1/A_1)$, then the composition of isogenies

$$\phi_{A_1} : E_1 \to E_1/A_1, \quad \psi : E_1/A_1 \to E_2/A_2, \quad \hat{\phi}_{A_2} : E_2/A_2 \to E_2,$$

where ψ is an $\mathbb{F}_{p^2}$-isomorphism and $\hat{\phi}_{A_2}$ denotes the dual of ϕ_{A_2}, is the desired degree-ℓ^e isogeny from E_1 to E_2. The worst-case time complexity of MITM-basic is $T_1 = 2N$, where a unit of time is a degree-$\ell^{e/2}$ Vélu isogeny computation (cf. Remark 1). The average-case time complexity is $1.5N$. The attack has space complexity N.

3.2 Depth-First Search

The set of pairs $(j(E/A), A)$, with A ranging over all order-$\ell^{e/2}$ subgroups of $E[\ell^e]$, can also be generated by using a depth-first search (DFS) to traverse the tree in the left of Fig. 1 (and also the tree in the right of Fig. 1). We denote this variant of the meet-in-the-middle attack by MITM-DFS. We describe the depth-first search for $\ell = 2$.[2]

Let $\{P, Q\}$ be a basis for $E[2^{e/2}]$. Let $R_0 = 2^{e/2-1}P$, $R_1 = 2^{e/2-1}Q$, $R_2 = R_0 + R_1$ be the order-2 points on E. For $i = 0, 1, 2$, the degree-2 isogenies $\phi_i : E \to E_i = E/\langle R_i \rangle$ are computed, as are bases $\{P_0 = \phi_0(P), Q_0 = \phi_0(2Q)\}$,

[2] For the sake of concreteness, all implementation reports of CSSI attacks in this paper are for the case $\ell = 2$. However, all conclusions about the relative efficiencies of classical and quantum CSSI attacks for $\ell = 2$ are also valid for the $\ell = 3$ case.

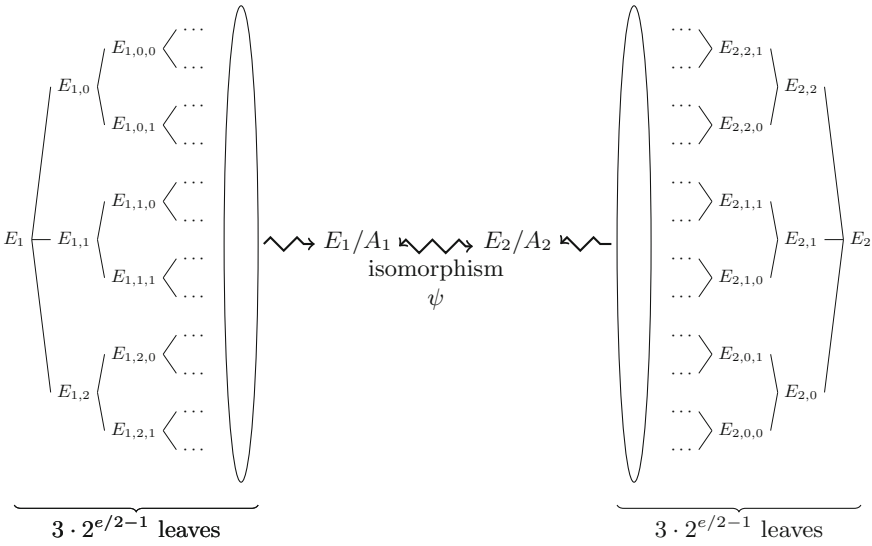

Fig. 1. Meet-in-the-middle attack for degree-2 isogeny trees.

$\{P_1 = \phi_1(Q), Q_1 = \phi_1(2P)\}$, $\{P_2 = \phi_2(P+Q), Q_2 = \phi_2(2P)\}$ for $E_0[2^{e/2-1}]$, $E_1[2^{e/2-1}]$, $E_2[2^{e/2-1}]$, respectively. A memory stack is initialized with the tuples $(E_0, 0, P_0, Q_0)$, $(E_1, 1, P_1, Q_1)$, $(E_2, 2, P_2, Q_2)$, and the tuple on the top of the stack is processed recursively as described next.

Suppose that we have to process (E_x, x, P_x, Q_x), where $x \in \{0,1,2\} \times \{0,1\}^{n-1}$ and $1 \leq n \leq e/2 - 1$. Let $B_0 = 2^{e/2-n-1}P_x$, $B_1 = 2^{e/2-n-1}Q_x$ and $B_2 = B_0 + B_1$ be the order-2 points on E_x. Let $R_{x0} = B_0$ and $R_{x1} = B_2$ (B_1 is the backtracking point), and compute the degree-2 isogenies $\phi_{xi} : E_x \to E_{xi} = E_x/\langle R_{xi} \rangle$ for $i = 0, 1$. Then two cases arise:

(i) If $n < e/2 - 1$, then let $P_{x0} = \phi_{x0}(P_x)$, $Q_{x0} = \phi_{x0}(2(P_x + Q_x))$, $P_{x1} = \phi_{x1}(P_x + Q_x)$, $Q_{x1} = \phi_{x1}(2P_x)$; one can check that $\{P_{xi}, Q_{xi}\}$ is a basis for $E_{xi}[2^{e/2-n-1}]$ for $i = 0, 1$. Then, $(E_{x1}, x1, P_{x1}, Q_{x1})$ is added to the stack and $(E_{x0}, x0, P_{x0}, Q_{x0})$ is processed next.

(ii) If $n = e/2 - 1$, the leaves $(j(E_{x0}), x0)$ and $(j(E_{x1}), x1)$ of the tree are stored in the table. If the stack is non-empty, then its topmost entry is processed next; otherwise the computation terminates.

The cost of building each of the two depth-first search trees is approximately $2N$ degree-2 isogeny computations, $2N$ degree-2 isogeny evaluations, $N/2$ point additions, and $2N$ point doublings (where $N = 3 \cdot 2^{e/2-1}$).

In contrast, the cost of building the table in MITM-basic (with $\ell = 2$) is approximately $\frac{Ne}{2}$ 2-isogeny computations, $\frac{Ne}{4} \log_2 \frac{e}{2}$ 2-isogeny evaluations, and $\frac{Ne}{4} \log_2 \frac{e}{2}$ point doublings (cf. Remark 1). A count of $\mathbb{F}_{p^2}$ multiplications and squarings yields the following costs for the core operations when Jacobian coordinates are used for elliptic curve arithmetic, isogeny computations,

and isogeny evaluations: 8 (2-isogeny computation), 12 (2-isogeny evaluation), 14 (point addition), 9 (point doubling). This gives a per-table cost of approximately $5.25Ne\log_2 e$ for MITM-basic, and a cost of $65N$ for MITM-DFS. Thus, the depth-first search approach yields a speedup by a factor of approximately $\frac{e}{12.4}\log_2 e$.

3.3 Implementation Report

The MITM-basic and MITM-DFS attacks (for $\ell = 2$) were implemented in C, compiled using gcc version 4.7.2, and executed on an Intel Xeon processor E5-2658 v2 server equipped with 20 physical cores and 256 GB of shared RAM memory.[3] We used fopenmp for the parallelization.

For $p = 2^{e_A}3^{e_B}d - 1$, the elliptic curve $E/\mathbb{F}_p : Y^2 = X^3 + X$ has $\#E(\mathbb{F}_p) = p+1$ and $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. A point $P \in E(\mathbb{F}_{p^2})$ of order $2 \cdot 3 \cdot d$ was randomly selected, and the isogenous elliptic curve $E_1 = E/\langle P\rangle$ was computed. Then, a random order-2^{e_A} subgroup A of $E_1(\mathbb{F}_{p^2})$ was selected, and the isogenous elliptic curve $E_2 = E_1/A$ was computed. Our CSSI challenge was to find a generator of A given E_1 and E_2.

We used Jacobian coordinates for elliptic curve arithmetic, isogeny computations, and isogeny evaluations. For MITM-basic, the leaves of the E_1-rooted tree shown in Fig. 1 were generated as follows. Let $\{P, Q\}$ be a basis for $E_1[2^{e/2}]$. Then for each pair $(b, k) \in \{0, 1, 2\} \times \{0, 1, \ldots, 2^{e/2-1} - 1\}$, triples

$$\begin{aligned}\left(j(E_1/\langle P + (b2^{e/2-1} + k)Q\rangle),\ \ b,\ \ b2^{e/2-1} + k\right), &\quad \text{for } b = 0, 1,\\ \left(j(E_1/\langle (2k)P + Q\rangle),\ \ b,\ \ k\right), &\quad \text{for } b = 2,\end{aligned}$$

Table 1. Meet-in-the-middle attacks for finding a 2^{e_A}-isogeny between two supersingular elliptic curves over $\mathbb{F}_{p^2}$ with $p = 2^{e_A} \cdot 3^{e_B} \cdot d - 1$. For each p, 25 randomly generated CSSI instances were solved and the average of the results are reported. The 'expected time' and 'measured time' columns give the expected number and the actual number of degree-$2^{e_A/2}$ isogeny computations for MITM-basic. The space is measured in bytes.

e_A	e_B	d	MITM-basic				MITM-DFS
			Expected time	Space	Measured time	Clock cycles	Clock cycles
32	20	23	$2^{17.17}$	$2^{20.72}$	$2^{17.26}$	$2^{34.50}$	$2^{31.73}$
34	21	109	$2^{18.17}$	$2^{21.83}$	$2^{18.24}$	$2^{35.49}$	$2^{32.71}$
36	22	31	$2^{19.17}$	$2^{22.87}$	$2^{19.14}$	$2^{36.43}$	$2^{33.67}$
38	23	271	$2^{20.17}$	$2^{23.99}$	$2^{20.20}$	$2^{37.59}$	$2^{34.60}$
40	25	71	$2^{21.17}$	$2^{25.04}$	$2^{21.15}$	$2^{38.63}$	$2^{35.71}$
42	26	37	$2^{22.17}$	$2^{26.09}$	$2^{22.11}$	$2^{39.83}$	$2^{36.78}$
44	27	37	$2^{23.17}$	$2^{27.14}$	$2^{23.25}$	$2^{41.07}$	$2^{37.87}$

[3] Our code for the MITM-basic, MITM-DFS and VW golden collision search CSSI attacks is available at https://github.com/JJChiDguez/CSSI.

were computed and stored in 20 tables sorted by j-invariant (each of the 20 cores was responsible for generating a portion of the leaves). The 20 tables were stored in shared RAM memory.

MITM-DFS was executed using 12 cores. Each core was responsible for generating a portion of the leaves, and the 12 sets of leaves were stored in shared RAM memory. Table 1 shows the time expended for finding 2^e-isogenies for $e \in \{32, 34, 36, 38, 40, 42, 44\}$ with the MITM-basic and MITM-DFS attacks. These experimental results confirm the accuracy of the attacks' heuristic analysis.

4 Golden Collision Search

4.1 Van Oorschot-Wiener Parallel Collision Search

Let S be a finite set of cardinality M, and let $f : S \to S$ be an efficiently-computable function which we shall heuristically assume is a random function. The van Oorschot-Wiener (VW) method [20] finds a collision for f, i.e., a pair $x, x' \in S$ with $f(x) = f(x')$ and $x \neq x'$.

Define an element x of S to be *distinguished* if it has some easily-testable distinguishing property. Suppose that the proportion of elements of S that are distinguished is θ. For $i = 1, 2, \ldots$, the VW method repeatedly selects $x_{i,0} \in_R S$, and iteratively computes a sequence $x_{i,j} = f(x_{i,j-1})$ for $j = 1, 2, 3, \ldots$ until a distinguished element $x_{i,a}$ is encountered. In that event, the triple $(x_{i,a}, a, x_{i,0})$ is stored in a table sorted by first entry. If $x_{i,a}$ was already in the table, say $x_{i,a} = x_{i',b}$ with $i \neq i'$, then a collision has been *detected* (see Fig. 2). The two colliding table entries $(x_{i,a}, a, x_{i,0})$, $(x_{i',b}, b, x_{i',0})$ can then be used to find a collision for f by iterating the longer sequence (say the ith sequence) beginning at $x_{i,0}$ until it is the same distance from $x_{i,a}$ as $x_{i',0}$ is from $x_{i',b}$, and then stepping both sequences in unison until they collide (see Fig. 3).

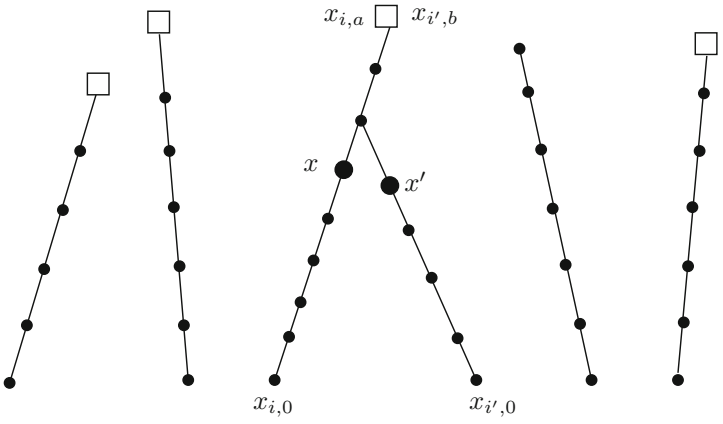

Fig. 2. VW method: detecting a collision (x, x').

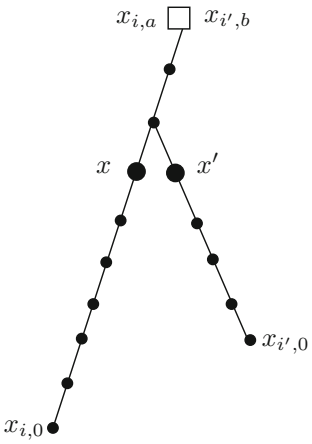

Fig. 3. VW method: finding a collision (x, x').

By the birthday paradox, the expected time before a collision occurs is $\sqrt{\pi M/2}$, where a unit of time is an f evaluation. After a collision has occurred, the expected time before it is detected is $1/\theta$, and thereafter the expected time to find the collision is approximately $3/\theta$. Thus, the expected time complexity of the VW method is approximately $\sqrt{\pi M/2} + 4/\theta$. The expected storage complexity is $\theta\sqrt{\pi M/2}$. The parameter θ can be selected to control the storage requirements.

The collision detecting stage of the VW method can be effectively parallelized. Each of the available m processors computes its own sequences, and the distinguished elements are stored in shared memory. The expected time complexity of parallelized VW is then $\frac{1}{m}\sqrt{\pi M/2} + \frac{2.5}{\theta}$. The space complexity is $\theta\sqrt{\pi M/2}$.

4.2 Finding a Golden Collision

A random function $f : S \to S$ is expected to have $(M-1)/2$ unordered collisions. Suppose that we seek a particular one of these collisions, called a *golden collision*; we assume that the golden collision can be efficiently recognized. Thus one continues generating distinguished points and collisions until the golden collision is encountered. The expected time to find q collisions is only about $\sqrt{q}$ times as much as that to find one collision. However, since not all collisions are equally likely and the golden collision might have a very low probability of detection (see [19]), it is necessary to change the version of f periodically.

Suppose that the available memory can store w triples $(x_{i,a}, a, x_{i,0})$. When a distinguished point $x_{i,a}$ is encountered, the triple $(x_{i,a}, a, x_{i,0})$ is stored in a memory cell determined by hashing $x_{i,a}$. If that memory cell was already occupied with a triple holding a distinguished point $x_{i',b} = x_{i,a}$, then the two triples are used to locate a collision.

Van Oorschot and Wiener proposed setting

$$\theta = \alpha\sqrt{w/M} \tag{1}$$

and using each version of f to produce βw distinguished points. Experimental data presented in [20] suggested that the total running time to find the golden collision is minimized by setting $\alpha = 2.25$ and $\beta = 10$. Then, for $2^{10} \leq w \leq M/2^{10}$, the expected running time to find the golden collisions when m processors are employed is slightly overestimated as

$$\frac{1}{m}(2.5\sqrt{M^3/w}). \tag{2}$$

Remark 3 (*verifying the VW heuristic analysis*). The running time estimate (2) relies on several heuristics, the most significant of which is that when $2^{10} \leq w \leq M/2^{10}$ then each version of f generates approximately $1.3w$ collisions, of which approximately $1.1w$ are distinct. The numbers $1.3w$ and $1.1w$ were determined experimentally in [20]. Then the probability that a particular version of f yields the golden collision is approximately $1.1w/(M/2)$, whence the expected number of function versions needed to locate the golden collision is approximately $0.45M/w$, and the expected total time is

$$0.45\frac{M}{w} \times 10w \times \frac{1}{2.25}\sqrt{M/w} \;\approx\; 2\sqrt{M^3/w}.$$

Table 2. Observed number $c_1 w$ of collisions and number $c_2 w$ of distinct collisions per version v of the MD5-based random function $f_{n,v} : \{0,1\}^n \to \{0,1\}^n$. The numbers are averages for 20 function versions when $w \leq 2^8$ and 10 function versions when $w \geq 2^9$.

w	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
$M = 2^{20}$																	
c_1	1.66	1.30	1.48	1.30	1.48	1.38	1.28	1.27	1.29	1.27	1.28	1.27	1.24	1.18	1.08	—	—
c_2	1.31	1.14	1.26	1.11	1.22	1.15	1.08	1.05	1.03	1.02	1.03	1.00	0.94	0.83	0.61	—	—
$M = 2^{24}$																	
c_1	1.38	1.36	1.38	1.37	1.33	1.31	1.31	1.36	1.32	1.33	1.31	1.30	1.30	1.29	1.29	1.27	1.24
c_2	1.21	1.14	1.16	1.16	1.12	1.10	1.11	1.13	1.11	1.11	1.09	1.06	1.06	1.05	1.04	1.00	0.95
$M = 2^{28}$																	
c_1	1.09	1.21	1.33	1.35	1.36	1.35	1.30	1.34	1.32	1.34	1.33	1.34	1.33	1.32	1.31	1.31	1.30
c_2	0.98	1.06	1.10	1.15	1.15	1.12	1.09	1.12	1.12	1.13	1.12	1.13	1.12	1.10	1.08	1.07	1.07
$M = 2^{32}$																	
c_1	1.21	1.44	1.35	1.35	1.35	1.31	1.30	1.32	1.33	1.35	1.33	1.34	1.33	1.34	1.33	1.33	1.32
c_2	1.00	1.18	1.17	1.12	1.16	1.10	1.10	1.11	1.13	1.13	1.13	1.13	1.12	1.13	1.12	1.12	1.11
$M = 2^{36}$																	
c_1	1.34	1.31	1.29	1.32	1.38	1.34	1.31	1.32	1.35	1.32	1.33	1.34	1.33	1.33	1.33	1.33	1.33
c_2	1.10	1.10	1.08	1.13	1.16	1.13	1.11	1.10	1.13	1.12	1.12	1.13	1.13	1.13	1.13	1.13	1.13

To verify these numbers, we ran some experiments using a "random" function $f_{n,v} : \{0,1\}^n \to \{0,1\}^n$ (so $M = 2^n$), where v is a string identifying the function version, and $f_{n,v}(X)$ is defined to be the n most significant bits of MD5(v, X). Table 2 lists the numbers of collisions and distinct collisions that were found for different values of (n, w), confirming the $1.3w$ and $1.1w$ numbers reported in [20].

4.3 The Attack

Let $I = \{1, 2, \ldots, N\}$ and $S = \{1, 2\} \times I$. For $i = 1, 2$, let $\mathcal{A}_i$ denote the set of all order-$\ell^{e/2}$ subgroups of $E_i[\ell^e]$, define $f_i : \mathcal{A}_i \to R_i$ by $f_i(A_i) = j(E_i/A_i)$, and let $h_i : I \to \mathcal{A}_i$ be bijections. Let $g : R \to S$ be a random function. Finally, define $f : S \to S$ by

$$f : (i, x) \mapsto g(f_i(h_i(x))).$$

Then one can view f as a "random" function from S to S.

Recall that one expects there are unique order-$\ell^{e/2}$ subgroups A_1, A_2 of $E_1[\ell^e]$, $E_2[\ell^e]$, respectively, with $j(E_1/A_1) = j(E_2/A_2)$. Let $y_1 = h_1^{-1}(A_1)$ and $y_2 = h_2^{-1}(A_2)$. Then the collision for f that we seek is the golden collision $(1, y_1)$, $(2, y_2)$. Using m processors and w cells of memory, the VW method can be used to find this golden collision in expected time

$$\frac{1}{m}(2.5\sqrt{8N^3/w}) \approx 7.1p^{3/8}/(w^{1/2}m).$$

Remark 4 (*finding any collision vs. finding a golden collision*). The problem of finding a collision for a hash function $H : \{0,1\}^* \to \{0,1\}^n$ and the problem of computing discrete logarithms in a cyclic group $\mathcal{G}$ can be formulated as problems of finding a collision for a random function $f : S \to S$, where $\#S = 2^n$ for the first problem and $\#S = \#\mathcal{G}$ for the second problem (see [20]). For both formulations, *any* collision for f yields a solution to the original problem. Thus, letting $N = 2^n$ or $N = \#\mathcal{G}$, the problems can be solved using van Oorschot-Wiener collision search in time approximately

$$\frac{1}{m}N^{1/2}.$$

In contrast, the only formulation of CSSI as a collision search problem for $f : S \to S$ that we know requires one to find a *golden* collision for f. For this problem, the van Oorschot-Wiener algorithm has running time approximately

$$N^{3/2}/(w^{1/2}m).$$

4.4 Implementation Report

The VW attack (for $\ell = 2$) was implemented in C, compiled using gcc version 4.7.2, and executed on an Intel Xeon processor E5-2658 v2 server equipped with 20 physical cores and 256 GB of shared RAM memory. We used fopenmp for the

parallelization and openssl's MD5 implementation. The CSSI challenges were the same as the ones in Sect. 3.3.

Let $\{P_1, Q_1\}$, $\{P_2, Q_2\}$ be bases for $E_1[2^{e/2}]$, $E_2[2^{e/2}]$, respectively. Noting that $N = 3 \cdot 2^{e/2-1}$, we identify the elements of $I = \{1, 2, \ldots, N\}$ with elements of $I_1 \times I_2$ where $I_1 = \{0, 1, 2\}$ and $I_2 = \{0, 1, \ldots, 2^{e/2-1} - 1\}$. The bijections $h_i : I_1 \times I_2 \to \mathcal{A}_i$ for $i = 1, 2$ are defined by

$$h_i : (b, k) \mapsto \begin{cases} P_i + (b2^{e/2-1} + k)Q_i, & \text{if } b = 0, 1, \\ (2k)P_i + Q_i, & \text{if } b = 2. \end{cases}$$

Let $S = \{1, 2\} \times I_1 \times I_2$. For $n \in \{0, 1\}^{64}$, we let $g_n : R \to S$ be the function computed using Algorithm 1. We then define the version $f_n : S \to S$ of f by $(i, x) \mapsto g_n(f_i(h_i(x)))$.

Algorithm 1. The "random" function g_n

Require: $n \in \{0, 1\}^{64}$ and $j \in \mathbb{F}_{p^2}$.
Ensure: Output $c \in \{1, 2\}$, $b \in I_1$, $k \in I_2$.
1: $counter := 0$.
2: **repeat**
3: $h := \text{MD5}(1, j, n, counter)$.
4: Let h' be the $e/2 + 2$ least significant bits of h, and parse h' as (k, c, b), where k, c, b have bitlengths $e/2 - 1$, 1, and 2, respectively.
5: $counter := counter + 1$.
6: **until** $b \neq 11$
7: **return** $(c + 1, b, k)$.

We set $\theta = 2.25\sqrt{w/2N}$, where $w = 2^t$, and declare an element $X \in S$ to be distinguished if the integer formed from the 32 least significant bits of $\text{MD5}(2, X)$ is $\leq 2^{32}\theta$. If X is distinguished, then it is placed in memory cell s, where s is the integer determined by the t least significant bits of $\text{MD5}(3, X)$. If a distinguished point is not encountered after $10/\theta$ iterations, then that trail is abandoned and a new trail is formed.

Table 3 shows the time expended for finding 2^e-isogenies for $e \in \{32, 34, 36, 38, 40, 42, 44\}$ with the VW attack. These experimental results confirm the accuracy of the VW attack's heuristic analysis.

To gain further confidence that the VW attack's heuristic analysis is accurate for cryptographically-interesting CSSI parameters (e.g., $e = 256$), we ran some experiments to estimate the number of collisions and distinct collisions for functions f_n when $e = 50, 60, 70, 80$. The results, listed in Table 4, confirm the $1.3w$ and $1.1w$ estimates in [20].

Table 3. Van Oorschot-Wiener golden collision search for finding a 2^{e_A}-isogeny between two supersingular elliptic curves over $\mathbb{F}_{p^2}$ with $p = 2^{e_A} \cdot 3^{e_B} \cdot d - 1$. For each p, the listed number of CSSI instances were solved and the median and average of the results are reported. The $\#f_n$'s column indicates the number of random functions f_n that were tested before the golden collision was found. The expected and measured times list the number of degree-$2^{e_A/2}$ isogeny computations.

e_A	e_B	d	w	Expected time	Number of runs	Median			Average		
						# f_n's	Measured time	Clock cycles	# f_n's	Measured time	clock cycles
32	20	23	2^9	$2^{23.20}$	25	180	$2^{23.55}$	$2^{40.79}$	319	$2^{24.38}$	$2^{41.62}$
34	21	109	2^9	$2^{24.70}$	25	256	$2^{24.54}$	$2^{41.89}$	714	$2^{26.02}$	$2^{43.37}$
36	22	31	2^{10}	$2^{25.70}$	25	369	$2^{26.06}$	$2^{43.51}$	838	$2^{27.25}$	$2^{44.70}$
38	23	271	2^{11}	$2^{26.70}$	25	196	$2^{26.15}$	$2^{43.70}$	567	$2^{27.69}$	$2^{45.23}$
40	25	71	2^{11}	$2^{28.20}$	25	162	$2^{26.36}$	$2^{43.99}$	1015	$2^{29.01}$	$2^{46.64}$
42	26	37	2^{12}	$2^{29.20}$	25	477	$2^{28.92}$	$2^{46.52}$	1940	$2^{30.95}$	$2^{48.55}$
44	27	37	2^{13}	$2^{30.20}$	25	431	$2^{29.78}$	$2^{47.46}$	942	$2^{30.91}$	$2^{48.58}$

Table 4. Observed number $c_1 w$ of collisions and number $c_2 w$ of distinct collisions per CSSI-based random function f_n. The numbers are averages for 25 function versions (except for $(e, w) \in \{(80, 2^{12}), (80, 2^{14}), (80, 2^{16})\}$ for which 5 function versions were used).

e	p	w	2^8	2^{10}	2^{12}	2^{14}	2^{16}
50	$2^{50}3^{31}179 - 1$	c_1	1.37	1.36	1.37	1.41	1.49
		c_2	1.14	1.12	1.12	1.11	1.09
60	$2^{60}3^{37}31 - 1$	c_1	1.37	1.34	1.34	1.35	1.36
		c_2	1.15	1.13	1.13	1.12	1.12
70	$2^{70}3^{32}127 - 1$	c_1	1.33	1.34	1.34	1.34	1.34
		c_2	1.13	1.14	1.13	1.13	1.13
80	$2^{80}3^{25}71 - 1$	c_1	1.35	1.32	1.33	1.34	1.33
		c_2	1.14	1.12	1.13	1.13	1.13

5 Comparisons

There are many factors that can affect the efficacy of an algorithm.

1. *Time*: the worst-case or average-case number of basic arithmetic operations performed by the algorithm.
2. *Space*: the amount of storage (RAM, hard disk, etc.) required.
3. *Parallelizability*: the speedup achievable when running the algorithm on multiple processors. Ideally, the speedup is by a factor equal to the number of

processors, and the processors do not need to communicate with each other; if this is the case then the parallelization is said to be *perfect*[4].

4. *Communication costs*: the time taken for communication between processors, and the memory access time for retrieving data from large storage devices. Memory access time can be a dominant cost factor when using extremely large storage devices [2].
5. *Custom-designed devices*: the possible speedups that can be achieved by executing the algorithm on custom-designed hardware. Examples of such devices are TWINKLE [24] and TWIRL [25] that were designed for the number field sieve integer factorization algorithm.

In this section we analyze and compare the efficacy of the meet-in-the-middle algorithm, VW golden collision search, and a mesh sorting algorithm for solving CSSI. We make two assumptions:

1. The number m of processors available is at most 2^{64}.
2. The total amount of storage w available is at most 2^{80} units.

Our analysis will ignore communication costs, and thus our running time estimates can be considered to be lower bounds on the "actual" running time.

Remark 5 (*feasible amount of storage and number of processors*). The Sunway TaihuLight supercomputer, the most powerful in the world as of March 2018, has $2^{23.3}$ CPU cores [29]. In 2013, it was estimated that Google's data centres have a total storage capacity of about a dozen exabytes[5] [29]. Thus it is reasonable to argue that acquiring 2^{64} processors and a storage capacity (with low access times) of several dozen yottabytes[6] for the purpose of solving a CSSI problem will be prohibitively costly for the foreseeable future.

5.1 Meet-in-the-Middle

As stated in Sect. 3, the running time of MITM-basic and MITM-DFS is approximately $2N$ and the storage requirements are N, where $N \approx p^{1/4}$. Since for $N \geq 2^{80}$ the storage requirements are infeasible, we deem the meet-in-the-middle attacks to be prohibitively expensive when $N \gg 2^{80}$.

Of course, one can trade space for time. One possible time-memory tradeoff is to store a table with entries $(j(E_1/A_1), A_1)$, where A_1 ranges over a w-subset of order-$\ell^{e/2}$ subgroups of $E_1[\ell^e]$. Next, for each order-$\ell^{e/2}$ subgroup A_2 of $E_2[\ell^e]$, E_2/A_2 is computed and $j(E_2/A_2)$ is searched in the table. If no match is found, then the algorithm is repeated for a disjoint w-subset of order-$\ell^{e/2}$ subgroups of $E_1[\ell^e]$, and so on. The running time of this time-memory tradeoff is approximately

$$(w + N)\frac{N}{w} \approx N^2/w.$$

[4] If the processors share the same storage space, then frequent storage accesses might decrease the parallelizability of the algorithm.

[5] An exabyte is 2^{60} bytes.

[6] A yottabyte is 2^{80} bytes.

For MITM-basic, the unit of time is an $\ell^{e/2}$-isogeny computation. For MITM-DFS, the running time (for $\ell = 2$) can be scaled to $\ell^{e/2}$-isogeny computations by dividing by $\frac{e}{12.4}\log_2 e$ (cf. Sect. 3.2). One can see that this time-memory-tradeoff can be parallelized perfectly.

Another possible time-memory tradeoff is to store $(j(E_1/A_1), A_1)$, where A_1 ranges over all order-ℓ^c subgroups of $E_1[\ell^e]$ and $c \approx \log_\ell w$. Let $d = e - c$. Then, for each order-ℓ^d subgroup A_2 of $E_2[\ell^e]$, E_2/A_2 is computed and $j(E_2/A_2)$ is searched in the table. One can check that the running time of this time-memory tradeoff is approximately N^2/w, and that it can be parallelized perfectly. Note that the unit of time here is an ℓ^d-isogeny computation instead of an $\ell^{e/2}$-isogeny computation. The larger tree of ℓ^d-isogenies can be traversed using a depth-first search; the running time is then the same as that of the MITM-DFS variant described in the previous paragraph.

5.2 Golden Collision Search

As stated in Sect. 4.3, the running time of van Oorschot-Wiener golden collision search is approximately

$$N^{3/2}/w^{1/2}.$$

The algorithm parallelizes perfectly.

5.3 Mesh Sorting

The mesh sorting attack is analogous to the one described by Bernstein [2] for finding hash collisions. Suppose that one has m processors arranged in a two-dimensional grid. Each processor only communicates with its neighbours in the grid. In one unit of time, each processor computes and stores pairs $(j(E_1/A_1), A_1)$, where A_1 is an order-$\ell^{e/2}$ subgroup of $E_1[\ell^e]$. Next, these stored pairs are sorted in time $\approx m^{1/2}$ (e.g., see [22]). In the next stage, a second two-dimensional grid of m processors computes and stores pairs $(j(E_2/A_2, A_2)$, where A_2 is an order-$\ell^{e/2}$ subgroup of $E_2[\ell^e]$, and the two sorted lists are compared for a match. This is repeated for a disjoint m-subset of order-$\ell^{e/2}$ subgroups A_2 until all order-$\ell^{e/2}$ subgroups of $E_2[\ell^e]$ have been tested. Then, the process is repeated for a disjoint subset of order-$\ell^{e/2}$ subgroups A_1 of $E_1[\ell^e]$ until a match is found. One can check that the calendar running time[7] is approximately

$$\left(m^{1/2} + m^{1/2}\frac{N}{m}\right)\frac{N}{m} \approx N^2/m^{3/2}.$$

5.4 Targetting the 128-Bit Security Level

The CSSI problem is said to have a 128-bit security level if the fastest known attack has total time complexity at least 2^{128} and feasible space and hardware costs.

[7] *Calendar time* is the elapsed time taken for a computation, whereas *total time* is the sum of the time expended by all m processors.

Suppose that $p \approx 2^{512}$, whereby $N \approx 2^{128}$; this would be a reasonable choice for the bitlength of p if the meet-in-the-middle attacks were assessed to be the fastest (classical) algorithm for solving CSSI. However, as noted above, the storage costs for the attacks are prohibitive. Instead, one should consider the time complexity of the time-memory tradeoffs, VW golden collision search, and mesh sorting under realistic constraints on the storage space w and the number m of processors. Table 5 lists the calendar time and the total time of these CSSI attacks for $(m, w) \in \{(2^{48}, 2^{64}), (2^{48}, 2^{80}), (2^{64}, 2^{80})\}$. One sees that in all cases the total time complexity is significantly greater than 2^{128}, even though we have ignored communication costs.

Table 5. Time complexity estimates of CSSI attacks for $p \approx 2^{512}$ and $p \approx 2^{448}$, and $\ell = 2$. All numbers are expressed in their base-2 logarithms. The unit of time is a $2^{e/2}$-isogeny computation.

	# processors m	space w	$p \approx 2^{512}$		$p \approx 2^{448}$	
			calendar time	total time	calendar time	total time
Meet-in-the-middle (DFS) time-memory tradeoff	48	64	138	186	106	154
	48	80	122	170	90	138
	64	80	106	170	74	138
Van Oorschot-Wiener golden collision search	48	64	112	160	88	136
	48	80	104	152	80	128
	64	80	88	152	64	128
Mesh sorting	48	—	184	232	152	200
	64	—	160	224	128	192

Since the total times for $p \approx 2^{512}$ in Table 5 are all significantly greater than 2^{128}, one can consider using smaller primes p while still achieving the 128-bit security level. Table 5 also lists the calendar time and the total time of these CSSI attacks for $(m, w) \in \{(2^{48}, 2^{64}), (2^{48}, 2^{80}), (2^{64}, 2^{80})\}$ when $p \approx 2^{448}$ and $N \approx 2^{112}$. One sees that all attacks have total time complexity at least 2^{128}, even though we have ignored communication costs. We can conclude that selecting SIDH parameters with $p \approx 2^{448}$ provides 128 bits of security against known classical attacks. For example, one could select the 434-bit prime

$$p_{434} = 2^{216}3^{137} - 1;$$

this prime is balanced in the sense that $3^{137} \approx 2^{217}$, thus providing maximal resistance to Petit's SIDH attack [21].

Remark 6 (*communication costs*). Consider the case $p \approx 2^{448}$, $e = 224$, $m = 2^{64}$, $w = 2^{80}$. From (1) and (2) we obtain $\theta \approx 1/2^{15.62}$ and an expected running time

of $2^{131.7}$. For each function version, the 2^{64} processors will generate approximately $2^{48.4}$ distinguished points per unit of time (i.e., a 2^{112}-isogeny computation). So, on average, the 2^{80} storage device will be accessed $2^{48.4}$ times during each unit of time. The cost of these accesses will certainly dominate the computational costs. Thus our security estimates, which ignore communication costs, should be regarded as being conservative.

5.5 Targetting the 160-Bit Security Level

Using similar arguments as in Sect. 5.4, one surmises that SIDH parameters with $p \approx 2^{536}$ offer at least 160 bits of CSSI security against known classical (see Table 6). For example, one could select the 546-bit prime

$$p_{546} = 2^{273}3^{172} - 1;$$

this prime is nicely balanced since $3^{172} \approx 2^{273}$.

Table 6. Time complexity estimates of CSSI attacks for $p \approx 2^{536}$ and $p \approx 2^{614}$, and $\ell = 2$. All numbers are expressed in their base-2 logarithms. The unit of time is a $2^{e/2}$-isogeny computation.

	# processors m	space w	$p \approx 2^{536}$		$p \approx 2^{614}$	
			calendar time	total time	calendar time	total time
Meet-in-the-middle (DFS) time-memory tradeoff	48	64	150	198	188	236
	48	80	134	182	172	220
	64	80	118	182	156	220
Van Oorschot-Wiener golden collision search	48	64	121	169	149	197
	48	80	113	161	141	189
	64	80	97	161	125	189
Mesh sorting	48	—	196	244	234	282
	64	—	172	236	210	274

5.6 Targetting the 192-Bit Security Level

Using similar arguments as in Sect. 5.4, one surmises that SIDH parameters with $p \approx 2^{614}$ offer at least 192 bits of CSSI security against known classical (see Table 6). For example, one could select the 610-bit prime

$$p_{610} = 2^{305}3^{192} - 1;$$

this prime is nicely balanced since $3^{192} \approx 2^{304}$.

5.7 Resistance to Quantum Attacks

The appeal of SIDH is its apparent resistance to attacks by quantum computers. What remains to be determined then is the security of CSSI against quantum attacks.

The fastest known quantum attack on CSSI is Tani's algorithm [26]. Given two generic functions $g_1 : X_1 \to Y$ and $g_2 : X_2 \to Y$, where $\#X_1 \approx \#X_2 \approx N$ and $\#Y \gg N$, Tani's quantum algorithm finds a claw, i.e., a pair $(x_1, x_2) \in X_1 \times X_2$ such that $g_1(x_1) = g_2(x_2)$ in time $O(N^{2/3})$. The CSSI problem can be recast as a claw-finding problem by defining X_i to be the set of all degree-$\ell^{e/2}$ isogenies originating at E_i, g_i to be the function that maps a degree-$\ell^{e/2}$ isogeny originating at E_i to the j-invariant of its image curve, and $Y = R$. Since $\#X_1 = \#X_2 = N \approx p^{1/4}$, this yields an $O(p^{1/6})$-time CSSI attack.

CSSI can also be solved by an application of Grover's quantum search [12]. Recall that if $g : X \to \{0, 1\}$ is a generic function such that $g(x) = 1$ for exactly one $x \in X$, then Grover's algorithm can determine the x with $g(x) = 1$ in quantum time $O(\sqrt{\#X})$. The CSSI problem can be recast as a Grover search problem by defining X to be the set of all ordered pairs (ϕ_1, ϕ_2) of degree-$\ell^{e/2}$ isogenies originating at E_1, E_2, respectively, and defining $g(\phi_1, \phi_2)$ to be equal to 1 if and only if the j-invariants of the image curves of ϕ_1 and ϕ_2 are equal. Since $\#X = N^2 \approx p^{1/2}$, this yields an $O(p^{1/4})$-time quantum attack on CSSI.

The Jao-De Feo paper [14] that introduced SIDH identified Tani's claw-finding algorithm as the fastest known attack, whether classical or quantum, on CSSI. The subsequent literature on SIDH used the simplified running time $p^{1/6}$ of Tani's algorithm (i.e., ignoring the implied constant in its $O(p^{1/6})$ running time expression) to select SIDH primes p for a desired level of security. In other words, in order to achieve a b-bit security level against known classical and quantum attacks, one selects an SIDH prime p of bitlength approximately $6b$. For example, the 751-bit prime $p = 2^{372}3^{239} - 1$ was proposed in [8] for the 128-bit security level, and this prime has been used in many subsequent works, e.g., [6,7,13,17,32]. Also, the 964-bit prime $p = 2^{486}3^{301} - 1$ was proposed in [13] for the 160-bit security level.

However, this assessment of SIDH security does not account for the cost of the $O(p^{1/6})$ quantum space requirements of Tani's algorithm, nor for the fact that Grover's search does not parallelize well—using m quantum circuits only yields a speedup by a factor of $\sqrt{m}$ and this speedup has been proven to be optimal [31]. Some recent work [1,16] suggests that Tani's and Grover's attacks on CSSI are costlier than the van Oorschot-Wiener golden collision search algorithm. If this is indeed the case, then one can be justified in selecting SIDH primes p_{434} (instead of p_{751}), p_{546} (instead of p_{964}) and p_{610} in order to achieve the 128-, 160- and 192-bit security levels, respectively, against both classical and quantum attacks. Furthermore, SIDH parameters with p_{434} could be deemed to meet the security requirements in NIST's Category 2 [18] (classical and quantum security comparable or greater than that of SHA-256 with respect to collision resistance), and p_{610} could be deemed to meet the security requirements in NIST's Category 4 [18] (classical and quantum security comparable to that of SHA-384).

5.8 SIDH Performance

A significant benefit of using smaller SIDH primes is increased performance. The reasons for the boost in SIDH performance are twofold. First, since the computation of the ground field $\mathbb{F}_p$ multiplication operation has a quadratic complexity, any reduction in the size of p will result in significant savings. Since high-end processors have a word size of 64 bits, the primes p_{751}, p_{546} and p_{434} can be accommodated using twelve, nine and seven 64-bit words, respectively. Hence, if $\mathbb{F}_p$ multiplication using p_{751} can be computed in T clock cycles, then a rough estimation of the computational costs for $\mathbb{F}_p$ multiplication using p_{434} and p_{546} is as low as $0.34T$ and $0.56T$, respectively. Second, since the exponents of the primes 2 and 3 in p_{434} and p_{546} are smaller than the ones in p_{751}, the computation of the isogeny chain described in Sect. 2.2 (see Remark 1) is faster.

Table 7 lists timings for SIDH operations for p_{434}, p_{546} and p_{751} using the SIDH library of Costello et al. [5]. The timings show that SIDH operations are about 4.8 times faster when p_{434} is used instead of p_{751}.

Table 7. Performance of the SIDH protocol. All timings are reported in 10^6 clock cycles, measured on an Intel Core i7-6700 supporting a Skylake micro-architecture. The "CLN + enhancements" columns are for our implementation that incorporates improved formulas for degree-2 and degree-3 isogenies from [6] and Montgomery ladders from [10] into the CLN library.

Protocol phase		CLN library [8]			CLN + enhancements		
		p_{751}	p_{434}	p_{546}	p_{751}	p_{434}	p_{546}
Key Gen.	Alice	35.7	7.51	13.20	26.9	5.3	10.5
	Bob	39.9	8.32	14.84	30.5	6.0	11.7
Shared secret	Alice	33.6	7.01	12.56	24.9	5.0	10.0
	Bob	38.4	7.94	14.35	28.6	5.8	11.5

6 Concluding Remarks

Our implementations of the MITM and golden collision search CSSI attacks are, to the best of our knowledge, the first ones reported in the literature. The implementations confirm that the performance of these attacks is accurately predicted by their heuristic analysis.

Our concrete cost analysis of the attacks leads to the conclusion that golden collision search is more effective that the meet-in-the-middle attack. Thus one can use 448-bit primes and 536-bit primes p in SIDH to achieve the 128-bit and 160-bit security levels against *known* classical attacks on the CSSI problem. We emphasize that these conclusions are based on our understanding of how to best implement these algorithms, and on assumptions on the amount of storage

and the number of processors that an adversary might possess. On the other hand, our conclusions are somewhat conservative in that the analysis does not account for communication costs. Moreover, whereas it is generally accepted that the AES-128 and AES-256 block ciphers attain the 128-bit security level in the classical and quantum settings, the time it takes to compute a degree-2^{112} isogeny (which is the unit of time for the golden collision search CSSI attack with balanced 448-bit prime p) is considerably greater than the time for one application of AES-128 or AES-256.

Acknowledgements. We thank Steven Galbraith for the suggestion to traverse the MITM trees using depth-first search. We also thank Sam Jaques for the many discussions on Grover's and Tani's algorithms.

References

1. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J., Menezes, A., Rodríguez-Henríquez, F.: On the cost or computing isogenies between supersingular elliptic curves. Cryptology ePrint Archive: Report 2018/313. http://eprint.iacr.org/2018/313
2. Bernstein, D.: Cost analysis of hash collisions: will quantum computers make SHARCS obsolete? In: Workshop Record of SHARCS 2009: Special-purpose Hardware for Attacking Cryptographic Systems (2009). https://cr.yp.to/papers.html#collisioncost
3. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 163–169. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054319
4. Charles, D., Goren, E., Lauter, K.: Cryptographic hash functions from expander graphs. J. Cryptol. **22**, 93–113 (2009)
5. Costello, C., et al.: SIDH Library. https://www.microsoft.com/en-us/research/project/sidh-library/
6. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 303–329. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_11
7. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 679–706. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_24
8. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21
9. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Math. Cryptol. **8**, 209–247 (2014)
10. Faz-Hernández, A., López, J., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. IEEE Trans. Comput. **67**, 1622–1636 (2018)

11. Galbraith, S.D., Petit, C., Silva, J.: Identification protocols and signature schemes based on supersingular isogeny problems. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 3–33. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_1
12. Grover, L.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual Symposium on Theory of Computing – STOC 1996. ACM Press, pp. 212–219 (1996)
13. Jao, D., et al.: Supersingular isogeny key encapsulation. Round 1 submission, NIST Post-Quantum Cryptography Standardization, 30 November 2017
14. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
15. Jao, D., Soukharev, V.: Isogeny-based quantum-resistant undeniable signatures. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 160–179. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_10
16. Jaques, S., Schanck, J.: Quantum cryptanalysis in the RAM model. Preprint (2018)
17. Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M.: Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA. In: Dunkelman, O., Sanadhya, S.K. (eds.) INDOCRYPT 2016. LNCS, vol. 10095, pp. 191–206. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49890-4_11
18. National Institute of Standards and Technology: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December 2016. https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf
19. van Oorschot, P.C., Wiener, M.J.: Improving implementable meet-in-the-middle attacks by orders of magnitude. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 229–236. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_18
20. van Oorschot, P., Wiener, M.: Parallel collision search with cryptanalytic applications. J. Cryptol. **12**, 1–28 (1999)
21. Petit, C.: Faster algorithms for isogeny problems using torsion point images. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 330–353. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_12
22. Schnorr, C., Shamir, A.: An optimal sorting algorithm for mesh connected computers. In: Proceedings of the Eighteenth Annual Symposium on Theory of Computing – STOC 1986. ACM Press, pp. 255–263 (1986)
23. Schoof, R.: Nonsingular plane cubic curves over finite fields. J. Comb. Theory Ser. A **46**, 183–211 (1987)
24. Shamir, A.: Factoring large numbers with the TWINKLE device. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 2–12. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_2
25. Shamir, A., Tromer, E.: Factoring large numbers with the TWIRL device. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 1–26. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_1
26. Tani, S.: Claw finding algorithms using quantum walk. Theor. Comput. Sci **410**, 5285–5297 (2009)
27. Vélu, J.: Isogénies entre courbes elliptiques. C. R. Acad. Sc. Paris **273**, 238–241 (1971)
28. Wikipedia: Sunway TaihuLight. https://en.wikipedia.org/wiki/Sunway_TaihuLight
29. Wikipedia: Exabyte. https://en.wikipedia.org/wiki/Exabyte#Google

30. Yoo, Y., Azarderakhsh, R., Jalali, A., Jao, D., Soukharev, V.: A post-quantum digital signature scheme based on supersingular isogenies. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 163–181. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_9
31. Zalka, C.: Grover's quantum searching algorithm is optimal. Phys. Rev. A **60**, 2746–2751 (1999)
32. Zanon, G.H.M., Simplicio, M.A., Pereira, G.C.C.F., Doliskani, J., Barreto, P.S.L.M.: Faster isogeny-based compressed key agreement. In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. LNCS, vol. 10786, pp. 248–268. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_12