Faster SeaSign signatures through improved rejection sampling

Thomas Decru[1], Lorenz Panny[2], and Frederik Vercauteren[1]

thomas.decru@kuleuven.be, lorenz@yx7.cc, frederik.vercauteren@kuleuven.be

[1] imec-COSIC, ESAT, KU Leuven, Belgium
[2] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, The Netherlands

Abstract. We speed up the isogeny-based "SeaSign" signature scheme recently proposed by De Feo and Galbraith. The core idea in SeaSign is to apply the "Fiat–Shamir with aborts" transform to the parallel repeated execution of an identification scheme based on CSIDH. We optimize this general transform by allowing the prover to *not* answer a limited number of said parallel executions, thereby lowering the overall probability of rejection. The performance improvement ranges between factors of approximately 4.4 and 65.7 for various instantiations of the scheme, at the expense of roughly doubling the signature sizes.

Keywords: Isogeny-based cryptography, signatures, SeaSign, rejection sampling, group actions.

1 Introduction

Elliptic curves have become a staple in various cryptographic applications in the past decades. In 1994, however, it was pointed out by Shor that a quantum computer could solve the *Discrete Logarithm Problem* (DLP), which is the core hardness assumption in elliptic-curve cryptography, in polynomial time [12]. For that reason, some of the recent research has shifted towards isogeny-based cryptography. In essence, the underlying mathematical problem is to find an isogeny between two given elliptic curves over a finite field. According to current knowledge, this problem can generally be assumed to be hard, even with the possible advent of quantum computers in mind.

The first instances of isogeny-based cryptosystems were proposed by Couveignes in 1997 [2], including a non-interactive key exchange protocol. His paper was not published at that time, and the idea was independently rediscovered in 2006 by Rostovtsev and Stolbunov [11]. More recently, Jao and De Feo

[*] Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported in part by the Commission of the European Communities through the Horizon 2020 program under project number 643161 (ECRYPT-NET) and in part by the Research Council KU Leuven grants C14/18/067 and STG/17/019. Date of this document: 2018.12.12.

proposed the so-called *Supersingular Isogeny Diffie–Hellman* (SIDH) scheme in 2011 [7]. This key-exchange protocol is the basis for SIKE [6], which was submitted to the post-quantum standardization project led by NIST [10]. SIDH is inherently different from the scheme of Couveignes and Rostovtsev–Stolbunov, mostly due to the fact that the endomorphism rings of supersingular elliptic curves are noncommutative. However, in 2018, Castryck, Lange, Martindale, Panny and Renes adapted the Couveignes–Rostovtsev–Stolbunov scheme to supersingular elliptic curves, which yields big efficiency improvements, and named the resulting protocol "CSIDH" [1]. In essence, this variation is made possible by restricting the family of curves under consideration to supersingular elliptic curves *defined over* $\mathbb{F}_p$ instead of $\mathbb{F}_{p^2}$.

CSIDH's small key sizes prompted De Feo and Galbraith to transform it into a signature scheme called *SeaSign* in the same year [4]. The construction uses the *Fiat–Shamir with aborts* framework, a technique commonly used in lattice-based cryptography [8], together with an isogeny-based identification scheme going back to Couveignes [2] and Rostovtsev–Stolbunov [11]. Their paper presents three different versions of SeaSign featuring various trade-offs between signature size, public-key size, and secret-key size. One of these versions attains 128 bits of security with signatures of less than one kilobyte. An issue impacting all of these schemes, however, is that the signing and verification times are rather substantial. Indeed, the basic SeaSign scheme takes (on average) almost two days to sign a message on a typical CPU, whereas the variants with smaller signatures or public keys still take almost ten minutes to sign (on average).

In this paper we tackle this performance issue in the more general setting of using group actions in a "Fiat–Shamir with aborts" scheme. We first discuss two (unfortunately mutually exclusive) adjustments that reduce the likelihood of rejections, which decreases the expected number of failed signing attempts before a success and hence makes signing more efficient. Next, we describe a modification that significantly speeds up the signing process at the cost of a small increase in signature size. The basic idea is to allow the prover to refuse answering a small fixed number of challenges, thereby reducing the overall probability of aborting. To attain a given security level, the total number of challenges—and correspondingly the signature size—will be somewhat larger than for standard Fiat–Shamir with aborts. As an application of these general techniques, we analyze the resulting speed-up for the various versions of the SeaSign signature scheme. The improvement is most noticeable when applied to the basic scheme: the original signing cost goes down from almost two days to just over half an hour. The other two, more advanced variants are still sped up by a factor of four to roughly two minutes per signature. Even though this is still too slow for most (if not all) applications, it is a significant improvement over the state of the art, and the underlying ideas of these speed-ups might be useful for other cryptographic schemes as well.

Acknowledgements. We are thankful to Steven Galbraith for his observation about shorter signatures in Remark 2, and to Taechan Kim for pointing out an error in an earlier version of the script in Appendix A.

1.1. Notation. The notation $[a;b]$ denotes the integer range $\{a, ..., b\}$.

Fix $n \geq 1$. Throughout, we will consider a transitive action of the abelian group $\mathbb{Z}^n$ on a finite set X, with a fixed element $E_0 \in X$. We will assume that "short" vectors in $\mathbb{Z}^n$ are enough to reach "almost all" elements of X.[1] Moreover, we assume that the cost of computing the action $[\mathbf{v}]E$ of a vector $\mathbf{v} \in \mathbb{Z}^n$ on an element $E \in X$ is linear in the 1-norm $\|\mathbf{v}\|_1 = \sum_{j=1}^n |v_j|$ of $\mathbf{v}$. (We will argue in Section 2.1 that these assumptions are satisfied in the CSIDH setting.)

2 Preliminaries

A good introductory reference for the applications of elliptic-curve isogenies in cryptography are the lecture notes by De Feo [3].

2.1. CSIDH. Consider a supersingular elliptic curve E defined over $\mathbb{F}_p$, where p is a large prime. While the endomorphism ring $\operatorname{End}(E)$ of E over the algebraic closure of $\mathbb{F}_p$ is noncommutative, the ring $\operatorname{End}_{\mathbb{F}_p}(E)$ of endomorphisms defined over $\mathbb{F}_p$ is an order $\mathcal{O}$ in the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$.

The ideal class group of $\operatorname{End}_{\mathbb{F}_p}(E) = \mathcal{O}$ is the quotient of the group of fractional invertible ideals in $\mathcal{O}$ by the principal fractional invertible ideals in $\mathcal{O}$, and will be denoted $\operatorname{cl}(\mathcal{O})$. The group $\operatorname{cl}(\mathcal{O})$ acts on the set of $\mathbb{F}_p$-isomorphism classes of elliptic curves with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$ through isogenies. More specifically, when given an $\mathcal{O}$-ideal $\mathfrak{a}$ and an elliptic curve E with $\operatorname{End}_{\mathbb{F}_p}(E) = \mathcal{O}$, we define $[\mathfrak{a}]E$ as the codomain of the isogeny $\varphi_{\mathfrak{a}} \colon E \to E/\mathfrak{a}$ whose kernel is $\bigcap_{\alpha \in \mathfrak{a}} \ker \alpha$. This isogeny is well-defined and unique up to $\mathbb{F}_p$-isomorphism.

There are formulas for computing $[\mathfrak{a}]E$. However, for general $\mathfrak{a}$, this computation requires large field extensions and hence has superpolynomial time complexity. To avoid this, CSIDH restricts to ideals of the form $\mathfrak{a} = \prod_{i=1}^n \mathfrak{l}_i^{e_i}$, where all $\mathfrak{l}_i$ are prime ideals of small norm ℓ_i, and such that the action of $\mathfrak{l}_i$ can be computed entirely over the base field $\mathbb{F}_p$. The curve $[\mathfrak{a}]E$ can then be computed by chaining isogenies of degrees ℓ_i. In principle the cost of computing the action of $\mathfrak{l}_i$ is in $\Theta(\ell_i)$, but for small values of ℓ_i it is dominated by a full-size scalar multiplication, which is why assuming cost $|e_1| + \cdots + |e_n|$ for computing the action of $\prod_{i=1}^n \mathfrak{l}_i^{e_i}$, as mentioned in Section 1.1, comes close to the truth. (Moreover, in our setting, the $|e_i|$ are all identically distributed, hence the differences in costs between various ℓ_i disappear on average.)

The CSIDH group action is defined as follows.

Parameters. Integers $n \geq 1$, $B \geq 0$. A prime p of the form $4 \cdot \ell_1 \cdots \ell_n - 1$, with ℓ_i small distinct odd primes. The elliptic curve $E_0 \colon y^2 = x^3 + x$ over $\mathbb{F}_p$. Write X for the set of ($\mathbb{F}_p$-isomorphism classes of) elliptic curves over $\mathbb{F}_p$ with $\operatorname{End}_{\mathbb{F}_p}(E) = \mathcal{O} = \mathbb{Z}[\pi]$, where π is the $\mathbb{F}_p$-Frobenius endomorphism.

[1] In other words: The action of $\mathbb{Z}^n$ on X factors through the quotient $Q = \mathbb{Z}^n/S$, where $S \leq \mathbb{Z}^n$ is the stabilizer of any $E \in X$, and we assume that Q is "sufficiently" covered by "short" vectors in $\mathbb{Z}^n$ under the quotient map $\mathbb{Z}^n \twoheadrightarrow Q$.

Group action. A group element is represented[2] by a vector $(e_1, ..., e_n) \in \mathbb{Z}^n$ sampled uniformly random from $[-B; B]^n$, which defines the ideal $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$ with $\mathfrak{l}_i = \langle \ell_i, \pi - 1 \rangle$. A public element is represented by a single coefficient $A \in \mathbb{F}_p$, describing the curve $E_A \colon y^2 = x^3 + Ax^2 + x$. The result of the action of an ideal $\mathfrak{a}$ on a public element $A \in \mathbb{F}_p$, assuming that E_A has the right endomorphism ring $\mathcal{O}$, is the coefficient B of the curve $[\mathfrak{a}]E_A \colon y^2 = x^3 + Bx^2 + x$.

The security assumption of the group action is that it is essentially a black-box version of the group $\mathrm{cl}(\mathcal{O})$ on which anyone can efficiently act by translations. In particular, given two elliptic curves $E, E' \in X$, it should be hard to find an ideal $\mathfrak{a}$ of $\mathcal{O}$ such that $E' = [\mathfrak{a}]E$.

Notice that it is not clear in general that the vectors in $[-B; B]^n$ cover the whole group, or even a "large" fraction. Unfortunately, sampling uniformly random from $\mathrm{cl}(\mathcal{O})$ is infeasible for large enough parameters, since there is no known efficient way to compute the structure of $\mathrm{cl}(\mathcal{O})$ in that case. In fact, knowing the exact class group structure would be sufficient to obtain much more efficient signatures, since no rejection sampling would be required [4]. Under the right assumptions however, the elements represented by vectors in $[-B; B]^n$ are likely to cover a large fraction of the group as long as $(2B + 1)^n \geq \#\mathrm{cl}(\mathcal{O})$. The values suggested for (n, B) in [1] are $(74, 5)$, which aim to cover a group of size approximately 2^{256}. This results in group elements of 32 bytes, public elements of 64 bytes, and a performance of about 40 ms per group action computation. For more details, see [1].

As stated in Section 1.1, we will from now on abstract away the underlying isogeny-based constructions and work in the setting of the group $(\mathbb{Z}^n, +)$ acting on a finite set X.

2.2. SeaSign. SeaSign [4] is a signature scheme based on a sketch of an isogeny-based identification scheme by Couveignes [2] and Stolbunov [13], in combination with the "Fiat–Shamir with aborts" construction [8] from lattice-based cryptography to avoid leakage. The identification part of SeaSign works as follows. Note that our exposition differs from [4] for consistency with the following sections.

Parameters. Like CSIDH, and additionally integers $\delta \geq 1$ and $S \geq 2$.[3]

Keys. Alice's private key is a list $\underline{\mathbf{a}} = (\mathbf{a}^{(1)}, ..., \mathbf{a}^{(S-1)})$ of $S-1$ vectors sampled uniformly random from $[-B; B]^n \subseteq \mathbb{Z}^n$.

For $i \in \{1, ..., S-1\}$, write $E_i := [\mathbf{a}^{(i)}]E_0$, that is, the result of applying the group element represented by $\mathbf{a}^{(i)} \in \mathbb{Z}^n$; then Alice's public key is the list $[\underline{\mathbf{a}}]E_0 := (E_1, ..., E_{S-1})$ of her secret vectors applied to the starting element E_0.

This situation is summarized in Figure 1.

[2] Note this representation matches the assumptions in Section 1.1.

[3] Technically there is no reason for δ to be an integer: it is sufficient that $\delta \in \frac{1}{B} \cdot \mathbb{Z}$, but we will assume $\delta \in \mathbb{Z}$ throughout for simplicity.

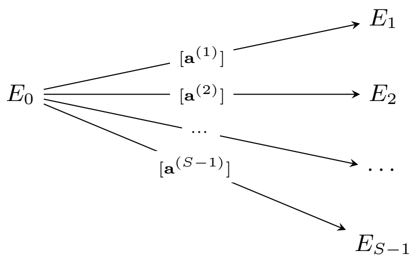

Figure 1. Structure of Alice's key pair.

Identification. Alice samples an ephemeral vector $\mathbf{b}$ uniformly random from the set $[-(\delta+1)B;\, (\delta+1)B]^n \subseteq \mathbb{Z}^n$. She then computes $E = [\mathbf{b}]E_0$ and commits to E. On challenge $c \in \{0, ..., S-1\}$, she computes $\mathbf{r} = \mathbf{b} - \mathbf{a}^{(c)}$ (where $\mathbf{a}^{(0)}$ is defined as $\mathbf{0}$). If $\mathbf{r} \in [-\delta B; \delta B]^n$, she reveals $\mathbf{r}$; else she rejects the challenge. Bob verifies that $[\mathbf{r}]E_c = E$.

See Figure 2 for a visual representation of this protocol.

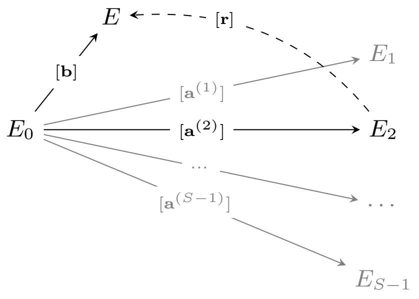

Figure 2. The identification scheme in the scenario $c = 2$.

Since an attacker (who cannot break the underlying isogeny problems) has a $1/S$ chance of winning, this identification scheme provides $\log_2 S$ bits of security. In order to amplify the security level, Alice typically computes $t \geq 1$ independent vectors $\mathbf{b}_1, ..., \mathbf{b}_t$ instead of just one. The verifier responds with t challenges $c_1, ..., c_t \in \{0, ..., S-1\}$. Alice then computes $\mathbf{r}_i = \mathbf{b}_i - \mathbf{a}^{(c_i)}$ for all $1 \leq i \leq t$ and reveals them if all of them are in $[-\delta B; \delta B]^n$; else she rejects the challenge. In order to not have to reject too often, δ must be rather large; more specifically, δ was chosen as nt in [4] to achieve a success probability of roughly $1/e$.

As mentioned in the introduction, [4] gives three SeaSign constructions. The original idea is the scheme above with $S = 2$, i.e., the public key is a single public element. This results in a large t and therefore a very large signature. The second scheme lets the number of private keys S range from 2 up to 2^{16}, which results in smaller, faster signatures at the expense of larger public-key sizes.[4] The final scheme reduced the size of the public key again by using a Merkle tree, at the cost of increasing the signature size. We will not elaborate on all those variants in detail.

To turn this identification scheme into a non-interactive signature protocol, the standard Fiat–Shamir transformation can be applied [5]. In essence, Alice obtains the challenges $c_1, ..., c_t$ herself by hashing the ephemeral public elements $[\mathbf{b}_1]E_0, ..., [\mathbf{b}_t]E_0$ together with her message. Alice then sends her signature $([\mathbf{b}_1]E_0, ..., [\mathbf{b}_t]E_0;\ \mathbf{r}_1, ..., \mathbf{r}_t)$ to Bob, who can recompute the challenges $c_1, ..., c_t$ to verify that indeed $[\mathbf{r}_i]E_{c_i} = [\mathbf{b}_i]E_0$ for all $i \in \{1, ..., t\}$.

3 The improved signature scheme

In this section we describe our improvements.

3.1. Core ideas.

1. The first improvement is minor (but still significant) and concerns the identification scheme itself: the following observations result in two variants of the scheme that are more efficient than the basic scheme.[5]

   – Variant $\mathcal{F}$: The ephemeral secret $\mathbf{b}$ is automatically independent of all secrets $\mathbf{a}^{(i)}$, hence can be revealed even if it lies outside of $[-\delta B; \delta B]^n$. We remark that this variant is described in [4] already but disregarded as only a single signing attempt is examined. When taking into account the average signing cost however, it can clearly improve performance, and we will quantify these improvements.

   – Variant $\mathcal{T}$: Depending on the entries of the concrete private keys $\mathbf{a}^{(i)}$, the ephemeral secret $\mathbf{b}$ can be sampled from a smaller set than the worst-case range used in SeaSign to reduce the probability of rejection. Indeed, although the j-th entry in each $\mathbf{a}^{(i)}$ is a priori sampled uniformly in $[-B; B]$, which gives rise to the interval $[-(\delta+1)B; (\delta+1)B]$ for the j-th coefficient of each ephemeral vector $\mathbf{b}$, it is obviously useless (since it will *always* be rejected) to sample the j-th coefficient outside the interval $[-\delta B + m_j;\ \delta B + M_j]$ with $m_j = \min\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$ and $M_j = \max\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$.

   It is clear that Variant $\mathcal{F}$ and Variant $\mathcal{T}$ are mutually exclusive: in Variant $\mathcal{T}$ the ephemeral secret $\mathbf{b}$ is sampled from a set that is dependent on the private keys $\mathbf{a}^{(i)}$, whereas for Variant $\mathcal{F}$ to work it is required that this sampling is done completely independently.

2. The second improvement is more significant and modifies the "Fiat–Shamir with aborts" transform as follows: assume the identification scheme uses s-bit challenges (corresponding to a probability of 2^{-s} that an attacker can

[4] In [4], S is always a power of 2, but any $S \geq 2$ works.

[5] The acronyms $\mathcal{F}$ and $\mathcal{T}$ refer to "**f**ull" and "**t**runcated" ranges, respectively.

cheat), and that each execution has probability of rejection ε. The SeaSign approach to attain security level λ is to simultaneously obtain $t = \lceil \lambda/s \rceil$ non-rejected executions of the identification protocol which happens with probability $(1-\varepsilon)^t$. Our approach increases the total number of challenges, but allows the prover to refuse answering a fixed number u of them, since this tolerates much higher rejection probabilities at the cost of a relatively small increase in public-key and signature size.

We now provide more details on each of the above ideas.

3.2. Identification scheme.

Parameters. Integers $S \geq 2$ and $\delta \geq 1$.

Keys. Like in SeaSign (Section 2.2).

Identification. Using Alice's key pair $(\underline{\mathbf{a}}, [\underline{\mathbf{a}}]E_0)$, a $(\log_2 S)$-bit identification protocol can be constructed as follows:

Variant $\mathcal{F}$	Variant $\mathcal{T}$
Alice samples a vector $\mathbf{b}$ uniformly random from the set ...	
$I = \left[-(\delta+1)B;\, (\delta+1)B\right]^n \subseteq \mathbb{Z}^n$.	$I = \prod_{j=1}^{n} \left[-\delta B + m_j;\, \delta B + M_j\right] \subseteq \mathbb{Z}^n$, where $m_j = \min\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$; $M_j = \max\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$.
She then computes $E = [\mathbf{b}]E_0$ and commits to E. On challenge $c \in \{0, ..., S-1\}$, she computes $\mathbf{r} = \mathbf{b} - \mathbf{a}^{(c)}$ (where $\mathbf{a}^{(0)}$ is defined as $\mathbf{0}$).	
If $c = 0$ or $\mathbf{r} \in [-\delta B; \delta B]^n$, ...	If $\mathbf{r} \in [-\delta B; \delta B]^n$, ...
... then she reveals $\mathbf{r}$; else she rejects the challenge. Bob verifies that $[\mathbf{r}]E_c = E$.	

Lemma 1. *The distribution of* <u>*revealed*</u> *vectors* $\mathbf{r}$ *is independent of* $\mathbf{a}^{(c)}$.

Proof. This is trivial in Variant $\mathcal{F}$ in the event $c = 0$. For the other cases, note that I is constructed such that $\mathbf{r} = \mathbf{b} - \mathbf{a}^{(c)}$ is uniformly distributed on a set containing $\Delta := [-\delta B; \delta B]^n$, no matter what $\mathbf{a}^{(c)}$ is. Therefore, the distribution of $\mathbf{r}$ conditioned on the event $\mathbf{r} \in \Delta$ is uniform on Δ independently of $\mathbf{a}^{(c)}$. $\square$

Remark 1. Lemma 1 only talks about the conditional distribution of $\mathbf{r}$ *if* it is revealed. Note that in Variant $\mathcal{T}$, the probability *that* it can be revealed is still correlated to the entries of $\mathbf{a}^{(c)}$, which may have security implications. We show in Section 3.3 how to get around this issue in a signature scheme.

3.3. Signature scheme. Our improved signature scheme is essentially the "Fiat–Shamir with aborts" construction also used in SeaSign (see Section 2.2), except that we allow the signer to reject a few challenges in each signature. The resulting scheme is parameterized by two integers $t \geq 0$, denoting the number of challenges the signer must answer correctly, and $u \geq 0$, the number of challenges she may additionally refuse to answer.

Write ID for (one of the variants of) the identification scheme in Section 3.2.

Keys. Alice's identity key consists of a key pair $(\underline{\mathbf{a}}, [\underline{\mathbf{a}}]E_0)$ as in ID.

Signing. To sign a message m, Alice first generates a list $\mathbf{b}_1, ..., \mathbf{b}_{t+u}$ of random vectors, each sampled as the vector $\mathbf{b}$ in ID. She computes the corresponding public elements $[\mathbf{b}_1]E_0, ..., [\mathbf{b}_{t+u}]E_0$ and hashes them together with the message m to obtain a list of challenges $c_1, ..., c_{t+u} \in \{0, ..., S-1\}$. To produce her signature, she then traverses the tuples $(\mathbf{b}_i, c_i)$ in a random order, computing the correct response $\mathbf{r}_i = \mathbf{b}_i - \mathbf{a}^{(c_i)}$ (as in ID) if possible and a rejection ✗ otherwise. Once t successful responses have been generated, the remaining challenges are all rejected in order not to leak any information about the rejection probability; cf. Remark 1.[6] Finally, the signature is

$$([\mathbf{b}_1]E_0, ..., [\mathbf{b}_{t+u}]E_0;\ \mathbf{r}_1, ..., \mathbf{r}_{t+u}),$$

where exactly u of the $\mathbf{r}_i$ equal ✗. (If less than t challenges could be answered, Alice aborts and retries the whole signing process with new values of $\mathbf{b}_i$.)

Verification. This again is standard: Bob first checks that at most u of the $t+u$ values $\mathbf{r}_i$ are ✗. He then recomputes the challenges $c_1, ..., c_{t+u}$ by hashing the message m together with the ephemeral elements $[\mathbf{b}_i]E_0$ and verifies that $[\mathbf{r}_i]E_{c_i} = [\mathbf{b}_i]E_0$ for all $i \in \{1, ..., t+u\}$ with $\mathbf{r}_i \neq$ ✗.

Remark 2. The signatures can be shortened further: Sending those $[\mathbf{b}_i]E_0$ with $\mathbf{r}_i \neq$ ✗ is wasteful. It is enough to send the hash H of all ephemeral elements $[\mathbf{b}_i]E_0$ instead, since Bob can extract c_i from H, recompute $[\mathbf{b}_i]E_0$ as $[\mathbf{r}_i]E_{c_i}$, and verify in the end that the hash H was indeed correct.

Remark 3. As mentioned earlier, one can reduce the public-key size by using a Merkle tree, but this does not significantly alter the computation time for any part of the protocol. Given that the main focus of our adjustments to SeaSign is speeding it up, we will therefore not investigate this avenue any further.

Security. The proof for the security for this scheme is completely analogous to the original SeaSign scheme. This follows from Lemma 1 and the fact that there are always a fixed number u of ✗ per signature in random positions. Instead of reproducing the proof here, we refer the reader to [4].

[6] This is why the tuples are processed in a random order: Proceeding sequentially and rejecting the remaining tail still leaks, since the number of ✗ at the end would be correlated to the rejection probability.

4 Analysis and results

In order to quantify our speed-ups compared to the original SeaSign scheme, we analyze our adjustments in the same context as [4]. This means that $(n, B) = (74, 5)$ and $\log_2 p \approx 512$. Furthermore we will require 128 bits of security and will let S range through powers of two between 2 and 2^{16}.

As mentioned before, Variant $\mathcal{F}$ and Variant $\mathcal{T}$ are mutually exclusive. For this reason, we computed the results for both cases to compare which performs better under given conditions. Variant $\mathcal{T}$ clearly converges to the original SeaSign scheme rapidly for growing S, while Variant $\mathcal{F}$ always keeps at least a little bit of advantage. It is clear that from a certain value of S onward, Variant $\mathcal{F}$ will always be better. For small S however, Variant $\mathcal{T}$ will outperform Variant $\mathcal{F}$ rather significantly for average-case key vectors.

We now discuss how to optimize the parameters (t, u, δ) for a given S. The main cost metric is the *expected* signing time[7]

$$\delta \cdot (t+u)/q\,,$$

where q is the probability of a full signing attempt being successful (i.e., at most u rejections ✗). This optimization problem depends on two random variables:

- The number Z of challenges that an *attacker* can successfully answer even though he cannot break the underlying isogeny problems.
- The number A of challenges that *Alice* can answer without leaking, i.e., the number of non-rejected challenges.

Since the $t + u$ challenges are independent, both Z and A are binomially distributed with count $t + u$. Let $T_{k,\alpha}$ denote the tail cumulative distribution function of $\mathrm{Bin}_{k,\alpha}$, i.e.,

$$T_{k,\alpha}(x) = \sum_{i=x}^{k} \binom{k}{i} \alpha^i (1-\alpha)^{k-i}\,,$$

which is the probability that a $\mathrm{Bin}_{k,\alpha}$-distributed variable attains a value of at least x. The success probability for an attacker is $1/S$, since he knows the correct answer to at most one of S challenges c. In order to achieve 128 bits of security, it is required that

$$\Pr[Z \geq t] = T_{t+u,1/S}(t) \leq 2^{-128}\,.$$

This condition implies that for fixed S and t, there is a maximal value $u_{max}(t)$ for u, the number of allowed rejections ✗, regardless of δ.

Let $\sigma(\delta)$ denote Alice's probability of being able to answer (i.e., not reject ✗) a single challenge for a given value of δ; hence $A \sim \mathrm{Bin}_{t+u,\sigma(\delta)}$. In order to find the optimal (u, δ) for a given t, we need to minimize the expression

$$\delta \cdot (t+u)/q(t, u, \delta)\,,$$

[7] Other optimizations could look at the sum of signing and verification time, or even take into account key generation time, but we will not delve into those options.

where

$$q(t, u, \delta) = \Pr[A \geq t] = T_{t+u,\sigma(\delta)}(t)$$

is the probability of a full signing attempt being successful. The function σ depends on the variant ($\mathcal{F}$ or $\mathcal{T}$). In case of Variant $\mathcal{F}$ we have

$$\sigma(\delta) = \frac{1}{S} + \frac{S-1}{S}\left(\frac{2\delta B + 1}{2(\delta+1)B+1}\right)^n.$$

For Variant $\mathcal{T}$, the function depends on the private keys in use. With fixed private keys $a^{(1)}, ..., a^{(S-1)}$ and the notation $m_j = \min\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$ and $M_j = \max\{0, a_j^{(1)}, ..., a_j^{(S-1)}\}$ as before, the formula becomes

$$\sigma(\delta) = \prod_{j=1}^{n} \frac{2\delta B + 1}{2\delta B + 1 - m_j + M_j}.$$

For our analysis we work with the expected probability over all possible keys.

Our results for the optimization problem can be found in Table 1. The `sage` [14] code that computes these values can be found in Appendix A; it takes about twelve minutes on a single core. We are quite confident that the values in Table 1 are optimal, but cannot strictly claim so since we have not *proven* that the conditions used in the script to terminate the search capture all optimal values, although this seems reasonable to assume.

There are two major differences in the way we present our data compared to [4]. First of all, we list the *expected* signing time instead of a single signing *attempt*, which represents the real cost more accurately. Second, we express the time in equivalents of "normal" CSIDH operations instead of in wall-clock time, which makes the results independent of a concrete choice of CSIDH implementation and eases comparison with other work.

Unsurprisingly, the biggest speed-up can be seen for the basic SeaSign scheme (i.e., $S = 2$), since that is where the largest δ could be found. The expected signing time is reduced by a factor of 65, whereas verification is sped up by a factor of roughly 31, at the cost of doubling the signature size. As predicted, Variant $\mathcal{F}$ outperforms Variant $\mathcal{T}$ from a certain point onward, which apparently is for $S \geq 2^4$. The case $S = 2^{16}$ gains a factor of 4.4 in the expected signing time and 6.0 in verification time. Note though that it only has 2.7% faster signing and 21% faster verification than the case $S = 2^{15}$ (which uses public keys half as big), which further emphasizes the importance of choosing the right trade-offs. Perhaps unsurprisingly, taking $u = u_{max}(t)$ often gives the best (expected) signing times, although this is not always the case: for instance, for $S = 2^{16}$ we have $u_{max}(10) = 29$, but $u = 22$ with a bigger δ yields (slightly) better results.

Table 1. Parameters for our improved SeaSign variants, optimizing for signing time. All of these choices provide ≥ 128 bits of security (of course assuming that the underlying isogeny problems are hard). Gray lines with variant "—" refer to the original parameter selection methodology suggested in [4]. The signature sizes make use of the observation in Remark 2. The "CSIDHs" columns express the computational load in terms of equivalents of a "normal" CSIDH operation, i.e., with exponents in $[-B; B]^n$, making use of the assumption that the cost is linear in the 1-norm of the input vector. Using current implementations [9,1], computing one "CSIDH"-512 takes approximately 40 ms of wall-clock time on a standard processor. Finally, the rightmost column shows the speed-up in signing and verification times compared to the original SeaSign scheme.

S	t	u	δ	Var.	Public-key bytes	Signature bytes	Expected signing attempts	Expected signing CSIDHs	Expected verifying CSIDHs	Speed-up factors
2^1	128	0	9472	—	64 b	19600 b	2.718	3295480	1212416	
2^1	337	79	114	$\mathcal{T}$	64 b	36838 b	1.058	50175	38418	65.7 | 31.6
2^2	64	0	4736	—	192 b	9216 b	2.718	823818	303104	
2^2	144	68	133	$\mathcal{T}$	192 b	18256 b	1.063	29962	19152	27.5 | 15.8
2^3	43	0	3182	—	448 b	5967 b	2.718	371862	136826	
2^3	83	56	141	$\mathcal{T}$	448 b	11695 b	1.078	21119	11703	17.6 | 11.7
2^4	32	0	2368	—	960 b	4320 b	2.718	205928	75776	
2^4	59	58	119	$\mathcal{F}$	960 b	9376 b	1.076	14985	7021	13.7 | 10.8
2^5	26	0	1924	—	1984 b	3442 b	2.717	135937	50024	
2^5	43	50	111	$\mathcal{F}$	1984 b	7301 b	1.085	11198	4773	12.1 | 10.5
2^6	22	0	1628	—	4032 b	2866 b	2.717	97322	35816	
2^6	33	42	108	$\mathcal{F}$	4032 b	5835 b	1.089	8824	3564	11.0 | 10.0
2^7	19	0	1406	—	8128 b	2440 b	2.717	72585	26714	
2^7	26	32	113	$\mathcal{F}$	8128 b	4550 b	1.107	7254	2938	10.0 | 9.1
2^8	16	0	1184	—	16320 b	2020 b	2.717	51469	18944	
2^8	22	30	106	$\mathcal{F}$	16320 b	4028 b	1.114	6139	2332	8.4 | 8.1
2^9	15	0	1110	—	32704 b	1883 b	2.717	45235	16650	
2^9	19	28	101	$\mathcal{F}$	32704 b	3609 b	1.121	5321	1919	8.5 | 8.7
2^{10}	13	0	962	—	65472 b	1609 b	2.717	33974	12506	
2^{10}	17	31	88	$\mathcal{F}$	65472 b	3593 b	1.113	4703	1496	7.2 | 8.4
2^{11}	12	0	888	—	131008 b	1473 b	2.716	28946	10656	
2^{11}	15	27	89	$\mathcal{F}$	131008 b	3155 b	1.126	4208	1335	6.9 | 8.0
2^{12}	11	0	814	—	262080 b	1340 b	2.716	24322	8954	
2^{12}	13	18	106	$\mathcal{F}$	262080 b	2413 b	1.165	3828	1378	6.4 | 6.5
2^{13}	10	0	740	—	524224 b	1207 b	2.716	20099	7400	
2^{13}	12	20	94	$\mathcal{F}$	524224 b	2436 b	1.153	3467	1128	5.8 | 6.6
2^{14}	10	0	740	—	1048512 b	1208 b	2.716	20099	7400	
2^{14}	11	19	92	$\mathcal{F}$	1048512 b	2276 b	1.157	3193	1012	6.3 | 7.3
2^{15}	9	0	666	—	2097088 b	1075 b	2.716	16279	5994	
2^{15}	10	15	100	$\mathcal{F}$	2097088 b	1934 b	1.191	2977	1000	5.5 | 6.0
2^{16}	8	0	592	—	4194240 b	944 b	2.716	12861	4736	
2^{16}	10	22	79	$\mathcal{F}$	4194240 b	2369 b	1.147	2898	790	4.4 | 6.0

References

1. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In *ASIACRYPT*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018. https://ia.cr/2018/383.
2. Jean Marc Couveignes. Hard homogeneous spaces., 1997. IACR Cryptology ePrint Archive 2006/291, https://ia.cr/2006/291.
3. Luca De Feo. Mathematics of isogeny based cryptography, 2017. https://defeo.lu/ema2017/poly.pdf.
4. Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions, 2018. IACR Cryptology ePrint Archive 2018/824. https://ia.cr/2018/824.
5. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.
6. David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. SIKE. Submission to [10]. http://sike.org.
7. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011. https://ia.cr/2011/506.
8. Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 598–616. Springer, 2009.
9. Michael Meyer and Steffen Reith. A faster way to the CSIDH, 2018. To appear at Indocrypt 2018. https://ia.cr/2018/782.
10. National Institute of Standards and Technology. Post-quantum cryptography standardization, December 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization.
11. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies, 2006. IACR Cryptology ePrint Archive 2006/145. https://ia.cr/2006/145.
12. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. https://arxiv.org/abs/quant-ph/9508027.
13. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. in Math. of Comm.*, 4(2):215–235, 2010.
14. The Sage Developers. *SageMath, the Sage Mathematics Software System (version 8.4)*, 2018. https://sagemath.org.

A Script for Table 1

```
#!/usr/bin/env sage
RR = RealField(1000)

secbits = 128
pbits = 512
csidhn, csidhB = 74, 5
isz = lambda d: 2*d*csidhB+1    # interval size
sigsize = lambda S, t, u, delta, var = 'O': ceil(1/8 * (0
      + ceil(min(t+u, u*log(t+u,2), t*log(t+u,2)))      # indices of rejections
      + ceil(log(S,2)*(t+u))                            # hash of ephemeral public keys
      + pbits*u                                         # rejected ephemeral public keys
      + t*ceil(log(isz(delta+(var=='F'))**csidhn,2))))  # revealed secret keys
pksize = lambda t, S: ceil(1/8 * (S-1)*pbits)


def Bin(n, p, k):      # Pr[ Bin_n,p >= k ]
    return sum(RR(1) * binomial(n, i) * p**i * (1-p)**(n-i) for i in range(k, n+1))

@cached_function
def joint_minmax_cdf(n, x, y, a, b):
    # Pr that min and max of n independent uniformly random
    # integers in [a;b] satisfy min <= x and max <= y.
    if x < a or y < a: return 0
    if y > b: y = b
    return RR((y-a+1)/(b-a+1))**n - (RR((y-x)/(b-a+1))**n if x < y else 0)

@cached_function
def joint_minmax(n, x, y, a, b):
    # Pr that min and max of n independent uniformly random
    # integers in [a;b] satisfy min = x and max = y.
    F = lambda xx, yy: joint_minmax_cdf(n, xx, yy, a, b)
    return F(x,y) - F(x-1,y) - F(x,y-1) + F(x-1,y-1)

def prob_accept_original(delta, S):
    # sample r from [-(delta+1)*B, (delta+1)*B];
    # reject r and a_c-r outside [-delta*B; +delta*B]
    return (isz(delta) / isz(delta+1)) ** csidhn # entries are independent

def prob_accept_full(delta, S):
    # sample r from [-(delta+1)*B, (delta+1)*B];
    # reject a_c-r outside [-delta*B; +delta*B]
    prob = (isz(delta) / isz(delta+1)) ** csidhn # entries are independent
    prob = 1/S*RR(1) + (S-1)/S*prob # can always reveal r
    return prob

def prob_accept_truncate(delta, S):
    prob = RR(0)
    for x in range(-csidhB, csidhB + 1):
        for y in range(x, csidhB + 1):
            # Pr[min and max coeffs of S-1 secret keys are x and y]
            weight = joint_minmax(S-1, x, y, -csidhB, +csidhB)
            # sample from [min(0,x)-delta*B, max(0,y)+delta*B];
            # reject outside [-delta*B; +delta*B]
            prob += weight * isz(delta) / (isz(delta) + max(0,y) - min(0,x))
    return prob ** csidhn # entries are independent

@cached_function
def max_u(t, S): # largest possible u for given S,t
    u, F = 1, lambda u: Bin(t+u, 1/S, t)
    while F(u) <= 2**-secbits: u *= 2
    lo, hi = u//2, u+1
    while hi - lo > 1:
        m = (lo+hi+1)//2
        if F(m) <= 2**-secbits: lo = m
        else: hi = m
    return lo
```

```
def prob_sign(t, u, sigma):
    return Bin(t+u, sigma, t)

def exp_csidhs_sign(t, u, delta, S, prob):
    pr_single = prob(delta, S)
    pr_all = prob_sign(t, u, pr_single)
    return (t+u) * delta / pr_all

def csidhs_verif(t, delta):
    return t * delta


for s in range(1, 17):
    S = 2**s

    t = ceil(secbits/log(S,2)) - 1
    last_umax = -1

    best_time, no_progress = 1./0, 0
    while True:

        if no_progress >= max(16, t/8): break #XXX hack
        t += 1

        if Bin(t + 4*t, 1/S, t) < 2**-secbits: umax = 4*t #XXX hack
        else: umax = max_u(t,S)

        no_progress_inner = True

        for variant in ('OTF' if t == ceil(secbits/log(S,2)) else 'TF'):

            for u in ([0] if variant == 'O' else reversed(range(last_umax+1, umax+1))):

                print >>sys.stderr, log(S,2), variant, t, u, no_progress

                prob = {'O': prob_accept_original,
                        'F': prob_accept_full,
                        'T': prob_accept_truncate}[variant]

                @cached_function
                def f(x): return exp_csidhs_sign(t, u, x, S, prob)

                if variant == 'O':
                    delta = csidhn * t
                else:
                    _, delta = find_local_minimum(f, 1, 2**24, tol=1)
                    delta = min((floor(delta), ceil(delta)), key = f)

                if f(delta) < best_time:
                    print ('logS={:2d} t={:3d} u={:3d} delta={:4d} {} ~> ' \
                            'pksize={:9,d}b sigsize={:7,d}b ' \
                            'tries={:8.6f} signCSIDHs={:9,d} verifCSIDHs={:9,d}') \
                            .format(log(S,2), t, u, delta, variant,
                                    pksize(t,S),
                                    sigsize(S, t, u, delta, variant),
                                    float(1 / prob_sign(t, u, prob(delta, S))),
                                    round(f(delta)),
                                    csidhs_verif(t, delta))
                    best_time = f(delta)
                    no_progress_inner = False

        no_progress = no_progress + 1 if no_progress_inner else 0

        last_umax = umax
```