On Fast Calculation of Addition Chains for Isogeny-Based Cryptography

Brian Koziel[1], Reza Azarderakhsh[2], David Jao[3], and Mehran Mozaffari-Kermani[4]

[1]Texas Instruments, kozielbrian@gmail.com.
[2]CEECS Dept and I-SENSE FAU, razarderakhsh@fau.edu.
[3]C&O Dept, U Waterloo, djao@uwaterloo.ca.
[4]EME Dept, RIT, mmkeme@rit.edu.

Abstract. Addition chain calculations play a critical role in determining the efficiency of cryptosystems based on isogenies on elliptic curves. However, finding a minimal length addition chain is not easy; a generalized version of the problem, in which one must find a chain that simultaneously forms each of a sequence of values, is NP-complete. For the special primes used in such cryptosystems, finding fast addition chains for finite field arithmetic such as inversion and square root is also not easy. In this paper, we investigate the shape of smooth isogeny primes and propose new methods to calculate fast addition chains. Further, we also provide techniques to reduce the temporary register consumption of these large exponentials, applicable to both software and hardware implementations utilizing addition chains. Lastly, we utilize our procedures to compare multiple isogeny primes by the complexity of the addition chains.

Keywords: Addition chains, post-quantum cryptography, isogeny-based cryptosystems, finite field

1 Introduction

An addition chain can be thought of as a sequence of integers starting from 1 to some number n, where each number is a sum of any two previous integers in the sequence. For finite fields, operations such as exponentiations, inversions, or square roots can be performed efficiently by utilizing an optimal addition chain, the smallest such addition chain sequence to reach n. In particular, fast exponentiation and inversion are paramount to the performance of scalar point multiplication in elliptic curve cryptography (ECC), pairings in pairing-based cryptosystems, and computing isogenies in the quantum-resistant isogeny-based cryptosystems [1].

There are several popular families of primes for fast computation of addition chains used in public key cryptography including Mersenne primes [2], Crandall primes [2], and Solinas primes [3]. Generally, these primes have a special form with most of the prime featuring all '1's. This speeds up most finite-field arithmetic tremendously and also produces extremely fast addition chains through

the use of a regular chain of squaring and multiplying by $2^s - 1$ for increasing values of s. Similarly, binary extension fields can take advantage of the Itoh-Tsujii [4] method to compute the large exponential for inversion which also utilizes towering values of $2^s - 1$, typically found in hardware implementations.

None of the above primes can be utilized for post-quantum cryptography based on isogenies on elliptic curves, primarily because the curves generated from these primes do not have many isogenies that are fast to compute. Therefore, in [1], a special shape of primes called smooth isogeny primes is presented that produce curves of smooth shape for fast isogeny computations. These are of the form $p = \ell_A^a \ell_B^b f \pm 1$, where ℓ_A and ℓ_B are relatively small primes, a and b are positive integers, and f is a small cofactor to make the number prime. Most primes of this form appear in the general prime category. However, if $\ell_A = 2$, then the second half of the prime is either all '1's in the case that the prime is minus 1 or all '0's in the case that the prime is plus 1. The all '0's form is much faster in terms of speed as it just requires squarings, but the all '1's pattern is still a regular structure that can take advantage of long chains of '1', similar to Solinas or Mersenne primes. If $\ell \neq 2$, then a basic windowing technique should be used, similar to the general primes. A majority of the known software [5,6,7,8] and hardware [9,10] implementations do not consider calculating fast addition chains, which can improve inversion and square root computations essentially for free.

Motivation. Current isogeny-based cryptography requires many exponentiations through the use of inversions and square roots. Many finite field inversions are required as points must be recovered from scalar point multiplications to compute isogenies between curves. Finite field square roots have also been introduced to create a basis for key compression [7,11]. For example, in the supersingular isogeny Diffie-Hellman key exchange protocol [6] with key compression [7], approximately 844 finite field inversions and 56 finite field square roots for 85-bit quantum security level were counted through test runs. Interestingly, [8] revised the SIDH formula to only require a constant 4 finite field inversions, making constant-time implementations feasible. Addition chains perform large exponentiations efficiently and in a constant set of operations. Thus, they prove both security and speed to exponentiations used in the inversion and square root operations.

In this paper, we study addition chains for primes used in post-quantum cryptography based on isogenies on supersingular elliptic curves. This cryptosystem resembles ECC with its use of point multiplications, but also provides quantum resistance by walking large degree isogeny graphs [1]. Our goal is to improve the speed and efficiency of addition chains used in isogeny-based cryptography so that implementation of this post-quantum scheme can be practical. Our contributions can be itemized as follows:

- We analyze the shape of smooth isogeny primes, which are applicable to post-quantum cryptography based on isogenies on supersingular elliptic curves, and present several methods to design fast addition chains.

Table 1. Notations used in this paper

Notation	Definition
$\mathbb{Z}$	The set of integers
$\mathbb{F}_{p^n}$	A finite field of size p^n
m	Power of 2 to represent families of special primes
k	Iterating over k bits at a time (as in k-ary method)
c	Optimal power of 2 for use in Hybrid Windowing Method
I, M, S, A	Inversion, Multiplication, Squaring, and Addition in $\mathbb{F}_p$
$\tilde{I}, \tilde{M}, \tilde{S}, \tilde{A}$	Inversion, Multiplication, Squaring, and Addition in $\mathbb{F}_{p^2}$

- We present a hybrid windowing method to optimize inversion for primes of the form $2^a \ell_B^b f \pm 1$.
- We present a windowing method with a subtraction to optimize computation of square root exponentials for $2^a \ell_B^b f - 1$.
- We introduce techniques to minimize the number of intermediate values that are stored for an addition chain.
- We present empirical results of our techniques on a few smooth isogeny primes.

2 Background of Addition Chains

In this section, we review basic concepts of addition chains, their computations, and a metric to compare them. All notations used in this paper are summarized in Table 1.

2.1 Addition Chains

We formally introduce addition chains with the following definitions. We point the reader to [12] for an in-depth explanation of addition chains.

Definition 1. *An addition chain is a sequence of integers* $(a_0, a_1, \ldots, a_r)$ *with* $a_0 = 1$ *and* $a_r = n$*, such that* $a_i = a_j + a_k$ *for any* $j, k < i$*.*

Definition 2. *An addition chain is optimal if its length is the smallest among all possible addition chains.*

We are interested in finding optimal or almost optimal addition chains. It has not been formally proven that finding an optimal addition chain is NP-complete, but finding the optimal addition chain sequence for multiple numbers is believed to be NP-complete.

Essentially, addition chains can be thought of as sums of preceding values in the sequence. This is analogous to exponentiation because multiplying two numbers with the same base is the same as adding the two exponentials, e.g. $x^i \times x^j = x^{i+j}$.

Algorithm 1 k-ary Precomputation

Input: n, k, c
Output: $c_i = c^i \bmod n$, with $i = 0 \dots 2^k - 1$
1. $c_0 = 1$
2. for i from 1 to $2^k - 1$ do
3. $c_i = (c_{i-1} \times c) \bmod n$
4. return c_i
5. end for

Algorithm 2 k-ary Exponentiation Method

Input: A, $c_i = c^i (i = 0 \dots 2^k - 1)$, $d = d_{b-1} d_{b-2} \dots d_1 d_0)_{2^k}$
Output: A^d
1. for i from $b - 1$ downto 0 do
2. $A = A^{2^k}$
3. $A = A \times c_{d_i}$
4. end for
5. return A

Definition 3. *A Brauer chain [13] is an addition chain that always uses the previous value for the next one. In other words, it is a sequence of integers* $(a_0, a_1, \dots, a_r)$ *with* $a_0 = 1$ *and* $a_r = n$*, such that* $a_i = a_j + a_{i-1}$.

Brauer chains utilize a stipulation that forces one of the inputs to be the previous value. This greatly reduces the number of possible combinations for addition chains. Several algorithms produce optimal Brauer chains, but unfortunately, these are typically not optimal among all addition chains. We point the readers to [13] for more analysis of Brauer chains. The general goal of Brauer chains is to precompute values and then use an accumulator to square and multiply these precomputed values.

2.2 Computations of Addition Chains

k-Ary Method. The binary method is among the simplest addition chains, that iterates over bits of an exponential with the square-and-multiply technique. However, this is part of a broader family, the k-ary method, which is also a form of a Brauer chain. The k-ary method iterates over k bits at a time by repeatedly performing k squarings followed by a multiplication with precomputed values. Algorithm 1 lists the precomputation phase and Algorithm 2 lists the iterative square-and-multiply method.

As an example, for $k = 5$ over a 512-bit exponential, there are approximately 511 squarings and 103 multiplications. Furthermore, at most 30 values must be

precomputed for the general case, for a grand total of 511 squarings and 133 multiplications.

Windowing Method. The sliding windowing method, presented in works such as [14], [15], [16], and [17], optimizes the k-ary method by breaking the exponential into specific windows up to a maximum of k bits. Efficient addition chain sequences are generated to satisfy each of these windows using methods such as Lucas chains, halving, approximation, and division. After that, these windows are applied when it is its turn as the exponential is squared many times. The main advantage of this over the standard k-ary method is that addition chain sequences are used to generate only the necessary windows efficiently to reduce the total number of multiplications and squarings.

2.3 Comparison of Addition Chains

For our purposes, we compare addition chains by the number of squarings and multiplications for the exponentiation, as well as the number of temporary registers that must be stored when implemented in hardware or software. For instance, it is interesting that the basic square-and-multiply, or binary method, requires many more multiplications than the windowing method for the general case, but only requires 2 registers. Indeed, this is among the slowest addition chains, but it is among the most space-efficient. For some implementations, squarings are faster than multiplications. For our purposes, we will also try to optimize for the relationship $S = 0.8M$.

2.4 Finite Field Inversion and Square Root

We are interested in using fast addition chains to compute the exponentiations needed by inversion and square root in $\mathbb{F}_{p^2}$. For any $A \in F_p$, finite field inversion computes a value $B = A^{-1}$ such that $A \cdot B = 1$, where $B \in F_p$. This can be computed using Fermat's little theorem, which holds that $A^{-1} = A^{p-2}$. Addition chains can be used to efficiently evaluate these large powers in a constant set of operations, to protect against timing attacks and simple power analysis attacks. Conversely, the extended Euclidean algorithm could be applied to obtain the inversion with a smaller time complexity, but at the cost of revealing some information about the operand.

For any $A \in F_p$, finite field square root computes a value $B = A^{1/2}$ such that $B \cdot B = A$ where $B \in F_p$. It should be noted that $-B \in \mathbb{F}_p$ is also a square root because $-B \cdot -B = A$ where $-B \in \mathbb{F}_p$. We utilize the approach given by [18] over even extension fields. The square root operation utilizes multiple exponentiations, $\frac{p-3}{4}$, $\frac{p-1}{2}$, and p in the case that $p \equiv 3 \bmod 4$ and $\frac{p-1}{4}$, $\frac{p-1}{2}$, and p in the case that $p \equiv 1 \bmod 4$. Notably, if $p \equiv 1 \bmod 4$ then there is an additional square root operation that is extremely expensive. The exponentiation by p in $\mathbb{F}_{p^2}$ is special in that it can be performed using the Frobenius operator [18], which only requires a finite field negation. This is shown in Equation 1. Consider an element, a, in $\mathbb{F}_{p^2}$ is represented as a_0 and a_1, where $a_0, a_1 \in \mathbb{F}_p$ and a_1 is the most significant element.

$$a^p = (a_0, a_1)^p = (a_0, -a_1) \tag{1}$$

In general, inversion requires a single exponentiation in $\mathbb{F}_p$ and the square root requires one or two exponentiations in $\mathbb{F}_{p^2}$. We refer the reader to [18] and the Appendix for a longer discussion of inversion and the square root in even extension fields.

3 Supersingular Isogeny Cryptosystems

This section serves as a brief review of supersingular isogeny theory and its application as a cryptosystem. We point the reader to [1] and [19] for a much more in-depth look at isogeny theory.

Isogeny-based cryptography relies on the difficulty to compute isogenies between elliptic curves. An isogeny between two elliptic curves, E_1 and E_2, is defined as a morphism $\phi : E_1 \to E_2$ such that $\phi(O) = O$ [19]. Essentially, this is a non-constant rational map between these two curves that preserves the null point. We are particularly interested in supersingular curves. The endomorphism ring of a curve is defined as as the ring of all isogenies from a curve to itself, under point addition and functional composition. A curve is considered supersingular if it features a endomorphism ring with $\mathbb{Z}$-rank equal to 4. Supersingular curves can be defined over $\mathbb{F}_{p^2}$ or $\mathbb{F}_p$. Therefore, a common field that includes all isogenous curves is $\mathbb{F}_{p^2}$. Supersingular curves have the property that for every prime $\ell \neq p$, there exist $\ell + 1$ isogenies of degree ℓ from a base curve. An isogeny can be computed over a kernel, κ, such that $\phi : E \to E/\langle\kappa\rangle$ by using Vélu's formulas [20].

The supersingular isogeny Diffie-Hellman (SIDH) key exchange protocol, for instance, operates similar to the standard Elliptic Curve Diffie-Hellman version. In this case, Alice and Bob both have private keys to perform a double point multiplication that spans the entire isogeny space. They compute isogenies over the agreed upon bases with their double point multiplication result as the kernel. They exchange these applied isogenies and perform a second set of double point multiplications and isogeny computations, to arrive on curves with the same j-invariant [1].

Key compression and decompression have been introduced for this key exchange protocol in [7]. In this revised protocol, each party deterministically creates a shared torsion basis, which is used to reconstruct some public information that was originally intended to be exchanged over a public channel in the standard protocol. The algorithms related to SIDH key compression were also recently improved in both speed and compression rate in [11]. An SIDH public key can be compressed to approximately $\frac{7}{2}\log p$ bytes [11].

How SIDH uses exponentiations. The SIDH protocol and compression were mentioned because they both use finite field inversions and square roots. Based on the new "projective" isogeny formulas presented in [8], 4 inversions are required for the SIDH protocol, far fewer than the original "affine" isogeny formulas. These inversions are necessary to recover the final curve coefficients,

Algorithm 3 Efficient Generation of Primes of the Form $2^a 3^b f - 1$

Output: Smooth isogeny primes of the form $2^a 3^b f - 1$
1. Define powers a and b for a balanced isogeny graph
2. Define a higher bound F on f
3. Define $\Pi = \prod_{i=1}^{k} p_i < F$, where p_i is a prime greater than 2 and 3 and Π is maximized
   3a. e.g. $\Pi = 5 \times 7 \times 11 \cdots$
4. Define the generator $g = (2^a 3^b)^{-1} \bmod \Pi$
5. While looking for primes, do
   6. Select some c_0 in $\mathbb{F}_\Pi^*$ //Must be coprime to each p_i
   7. While $(c_j \neq c_0)$, do //Test all candidates in cyclic sub-group for iteration j
      7a. Define $f = g + c \bmod \Pi$
      7b. Test if $p = 2^a 3^b f - 1$ is prime
      7c. $c_{j+1} = 3 \times c_j$ //Multiplication by 2 could also be used here
8. Return all valid primes p

basis points, and j-invariant in the SIDH algorithm. The strong compression algorithm in [7] deterministically finds coordinates that can be used as a torsion basis. One essential part to finding a torsion basis is ensuring that the points have the right order, which utilizes square roots in the curve equation to recover y-coordinates. It performs the square root at each iteration until it finds points that have the correct order.

Isogeny-based cryptosystems use primes of the form $\ell_A^a \ell_B^b \cdot f \pm 1$ where ℓ_A and ℓ_B are small primes, a and b are positive integers, and f is a small cofactor to make the number prime. This prime is used to define a supersingular elliptic curve, $E(\mathbb{F}_q)$ where $q = p^2$. In the literature, the fastest known isogeny computations are over $\ell_A = 2$ and $\ell_B = 3$, presented in [8]. For secure primes of this form, we want the relative size of ℓ_A^a and ℓ_B^b to be approximately equal for balanced isogeny graphs. Furthermore, these primes can fit nicely for software applications by making the size of the prime as close to a multiple of 32. Lastly, the quantum security under these schemes was shown to be approximately the number of bits divided by 6 in [1].

Efficiently Finding Smooth Isogeny Primes. From the prime number theorem in arithmetic progressions in [21], it can be shown that the density of such smooth isogeny primes is sufficient. A brute force approach could be used by testing all values of f, but we adapted the methods of [22] to greatly reduce the number of prime candidates of smooth isogeny forms. Algorithm 3 demonstrates the approach for primes of the form $2^a 3^b f - 1$, but simple changes to the generator in the algorithm make it applicable to other smooth isogeny primes. The algorithm ensures that all primes that are tested are already coprime to the product of all small primes used, Π. We further note that each candidate is coprime to 2 and 3 in our example.

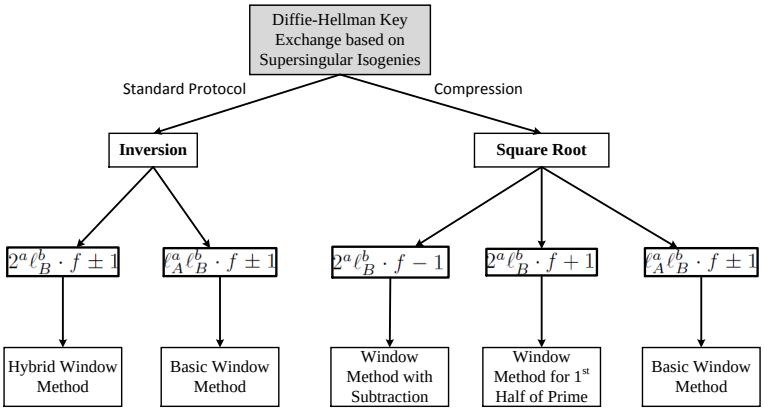

Fig. 1. Taxonomy of Addition Chains for Smooth Isogeny Prime Families

4 Fast Exponentiations for Smooth Isogeny Primes

In this section, we evaluate the structure of exponentiations for inversion and square root for smooth isogeny primes of the form $\ell_A^a\ell_B^b \cdot f \pm 1$ where ℓ_A and ℓ_B are small primes, a and b are positive integers, and f is a small cofactor to make the number prime. We break this prime form into the following families: $2^a\ell_B^b f - 1$, $2^a\ell_B^b f + 1$, and general smooth isogeny primes. Figure 1 summarizes our observations based on the addition chains method we found to be most effective.

4.1 $2^a\ell_B^b f - 1$ - Taking advantage of the least significant half of the prime

We introduce this family as a set of smooth isogeny primes that have the least signficant bits all set to '1'. Notably, these primes satisfy the Montgomery friendly property [23] to speed up Montgomery reduction [24]. Otherwise, it is essential to note that there will be many more factors of 2 than ℓ_B in the shape of the prime, so primes of this form are $p \equiv 3 \bmod 4$, equating to faster square root operations.

Proposition 1. *Very fast addition chains can be generated for primes of the form $2^a\ell_B^b f - 1$ by using an adaptation of the windowing method for the most significant half of the exponentials and precomputing a large value $2^c - 1$ for the least significant half.*

Proposition 1 is straightforward as the first half of primes of this form appear random and the second half is all '1's. For inversion, we are interested in fast addition chains for $p - 2$. For the square root, we want fast addition chains for $\frac{p-3}{4}$ and $\frac{p-1}{2}$. Luckily, for primes of the form $2^a\ell_B^b f - 1$, only the last few bits of these exponentials are different. Thus, fast addition chains are extremely similar among these exponentiations. We present the general procedure in Algorithm 4.

Algorithm 4 Hybrid windowing method for primes of the form $2^a \ell_B^b f - 1$.

Input: Smooth Isogeny prime of form $2^a \ell_B^b f - 1$
Output: Fast Addition Chains for $p-2, \frac{p-3}{4}$, p, and $\frac{p-1}{2}$
1. Split first half of prime into various max-sized windows
  1a. The best choice of window size varies based on the prime
2. Include an additional $2^c - 1$, such that c makes a large window of all '1's
that minimizes the number of multiplication windows for the second half of the
prime as well as minimizing the number of multiplications to generate
3. Determine good addition sequences to generate the windows
  3a. Add additional stored values if necessary
4. Slightly alter choice of c and multiplications to finish the addition chain between
$p-2, \frac{p-3}{4}$, p, and $\frac{p-1}{2}$

As Algorithm 4 shows, the general procedure starts by dividing up the first half of the prime. The size of the window depends on the shape of the first half of the prime, but is typically more than 7 bits for primes of this family of size 512-bits or larger. After the windows have been found, the addition chain sequences are constructed to encapsulate each of these windows. [14] provides one such algorithm to make addition chain sequences to generate these windows. However, we complete the sequence by using our own pivot judging, which we found to be very effective. This method determines which number acts as the best pivot. We judged potential candidates based on:

- Number of newly connected elements with the inclusion of the pivot
- Cost to generate the pivot value based on existing values (doubles are scored higher)
- Among high scoring pivots, the uniqueness of the connected elements are valued

Based on these criteria and the abundance of windows available, relatively few additional pivot values were added to complete the addition sequence. Our judging criteria prioritized values that could be obtained through squarings rather than multiplications, to reduce the total complexity of the addition chain sequence. Since all windows were found as odd numbers, starting and ending with a '1', primarily even values were added to finalize the addition sequence.

For the second half of the prime (a long chain of '1's), we require a high value $2^c - 1$. Typically, this value will be a few bits larger than the large window at the beginning of the exponential. The value $2^c - 1$ essentially acts as a very large window for the structured second half of the prime. This value generally fits nicely into the number of '1's at the end. The idea is to raise the value of c so that there are fewer windows on the second half of the prime. Indeed, larger values of $2^c - 1$ require more intermediate squarings and perhaps multiplications, but could reduce even more multiplications for the remaining windows. We consider

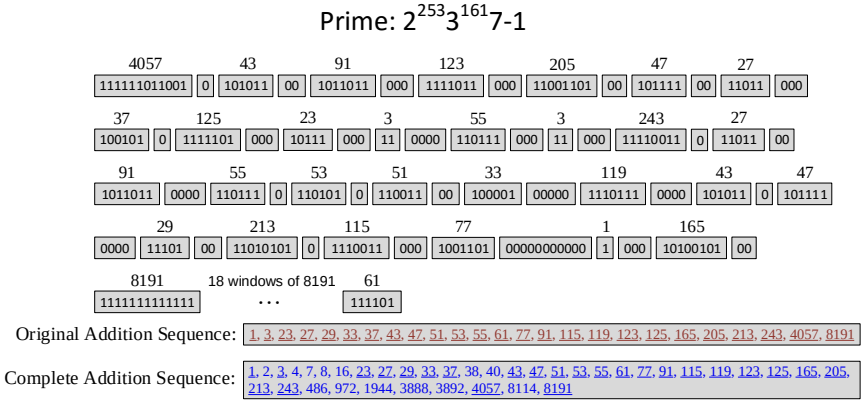

Fig. 2. Hybrid windowing method for $2^a \ell_B^b f - 1$

Table 2. Breakdown of Costs for Addition Chains for $2^{253}3^{161}7 - 3$

Operation	Cost
Window Generation	$28M + 9S$
Applying Windows $1st$ Half	$28M + 245S$
Applying Windows $2nd$ Half	$20M + 254S$
Total	$76M + 508S$

this a "hybrid" windowing method because there are different strategies for the first and second half of the prime.

4.2 $2^a \ell_B^b f - 1$ - An Illustrative Example - $2^{253}3^{161}7 - 1$

As an example of a prime in this field, we point to the prime for 85-bit quantum security presented in [1]. We want to create fast addition chains for inversion, or $2^{253}3^{161}7 - 3$. From the above strategy, we start by taking windows of the prime. We are looking at approximately 256 bits for this prime, so a max window size of about 7-10 bits is sufficient. In Figure 2, we use a max window of size 8, which was found to be the fastest addition chain based on our model. It was determined that the optimal value of c was 13, or $2^{13} - 1 = 8191$, which required only a single squaring and multiplication to reach and will complete the second half of the prime in 19 windows. Larger values of c would have required more multiplications and squarings, while not reducing the number of final windows enough to make it worth it. Table 2 illustrates the cost breakdown of various parts of the exponentiation for inversion.

4.3 $2^a \ell_B^b f - 1$ - Why use constant-time square roots?

Proposition 2. *For a non-constant time implementation of the square root, fast inversion algorithms such as the extended Euclidean algorithm can be used*

to produce negative values to greatly reduce the number of multiplications in an exponentiation.

To demonstrate Proposition 2, we point to the fact that key compression and decompression, which require the square root, only reconstruct information that would be transmitted over a public channel. Thus, as long as a fast inversion produces an addition chain requiring far fewer multiplications, its use may be justified. Typically, addition-subtraction chains are used for scalar point multiplication operations where the negative of a point is easy to obtain, such as in [25]. But the requirement of constant time for security in compression and decompression is not necessary and addition chains can benefit as a result.

As an example to this proposition, let us consider $p = 4091 = 111111111011_2$. We want to take the square root of an element, x, in $\mathbb{F}_{p^2}$ in a fast non-constant time fashion, so we produce the inverse, x^{-1}, using the Extended Euclidean Algorithm (EEA). For the first exponentiation, $x^{\frac{p-3}{4}}$, the exponential is $1111111110_2 = 2^{10} - 2 = 10000000000_2 - 10_2$. A standard binary method would require $8M + 9S$, but a standard binary method with the second representation would require $1I + 1M + 11S$. Thus, in the general case, if $I < 7M - 2S$, then the addition-subtraction chain method is faster. This serves as a toy example to show a possible way to speed up the square root exponentiations, and is also key to the non-adjacent form method (NAF) form of exponentiation [17].The NAF method does not necessarily mesh well with computing fast windows for addition chains because it typically iterates over single digits at a time and diminishes positive windows instead of growing them. However, it may provide far fewer multiplications for extremely long chains of '1's, which are prominent in square root exponentiations in the $2^a\ell_B^b f - 1$ family.

4.4 $2^a\ell_B^b f - 1$ - Addition-Subtraction Chains to Speed up Square Roots

Proposition 3. *Representing the exponentials used in square roots as $e-1$ with a final subtraction likely produces much faster addition chains.*

Proposition 3 alludes to using an addition-subtraction chain for fast exponentiation. The long chain of '1's in the least significant half of the exponentials can be avoided by using -1, or the original value's inverse. Let us consider exponentiating by p, for instance. This value can be rewritten as $p = (p + 1) - 1$. Thus, we can assume that we are exponentiating $2^a\ell_B^b f$, which has the second half all '0's. After we have found that exponential, we multiply by the inverse of the element and the exponentiation is complete.

For our example in Section 4.2, the second half of the prime required 1 extra squaring and 22 extra multiplications to generate and apply the final windows. We are essentially replacing this cost with a multiplication by the inverse. Thus, this method is faster if $\tilde{I} < \tilde{S} + 21\tilde{M}$ for this case. This may seem farfetched. However, in practice we have seen the ratio $\tilde{I} \approx 5\tilde{M}$ for 512-bit numbers in ARMv7 devices. This demonstrates that using a single subtraction at the end of

Algorithm 5 Windowing method with a subtraction $2^a\ell_B^b f - 1$.

Input: Smooth Isogeny prime of form $2^a\ell_B^b f - 1$
Output: Fast Addition Chains for $\frac{p-3}{4}$, p, and $\frac{p-1}{2}$
1. Add '1' to the prime to cancel out all of the second half of the prime
2. Split first half of result into various max-sized windows
   2a. The best choice of window size varies based on the prime
3. Determine small addition sequences to generate the windows
   3a. Add additional stored values if necessary
4. Perform a fast inversion using a method such as EEA
5. Perform a subtraction by multiplying by the inverse
6. Slightly alter final multiplications to finish the chains for $\frac{p-3}{4}$, p, and $\frac{p-1}{2}$

the addition chain saves the cost of $\tilde{S} + 16\tilde{M}$ in this case, most likely even more for larger prime sizes.

4.5 $2^a\ell_B^b \cdot f + 1$ Family

The $2^a\ell_B^b \cdot f + 1$ family features a prime shape with a long string of '0's. Thus, this can take advantage of a regular shape as well. Inversions within this family can be performed with the hybrid windowing method and the square root exponentiations feature a second half of the prime that is all '0's.

Exponentiation by $p-2$. The inversion exponentiation is similar to that of the $2^a\ell_B^b \cdot f - 1$ family, as the final half of the exponentation is all '1's. Thus, the hybrid windowing method is also valid for the $2^a\ell_B^b \cdot f + 1$ family and generates fast addition chains for these inversions.

Fast Exponentiations for Square Roots. This family has primes that are of the form $p \equiv 1 \bmod 4$. Thus, the exponentials $\frac{p-1}{4}$, $\frac{p-1}{2}$, and p are used in the square root process. This means that the windowing method can be used for the first half of these primes and the second half is simply squarings since it is all '0's. This exponentiation is similar to that of the addition-subtraction chains used in $2^a\ell_B^b \cdot f - 1$, but without the need to compute a fast inverse. However, these fast square root exponentations do not make up for the fact that an extremely expensive square root [18] in the field $\mathbb{F}_p$ must be performed. Thus, this family is not necessarily a good fit for key compression and decompression.

4.6 General Smooth Isogeny Family

The binary representation of digits used in today's processors means that the representation of other "general" smooth isogeny primes will appear pseudo-random since there are no powers of 2. The general isogeny primes can further be classified based on their form of their square root functions. The two classifications are $p \equiv 3 \bmod 4$ or $p \equiv 1 \bmod 4$. Clearly, these are the two groupings because otherwise the number would not be prime. In either case, the inversion

Algorithm 6 Minimizing register usage in windowing method

Input: Addition chain sequence based on the windowing method
Output: Efficient paths to perform the exponentiation with a reduced number of registers
1. Based on the addition sequence, generate a short path from 1 to the value of the first window
2. Remove values that are stored in registers based on the following criteria:
  2a. If a register has been used and is no longer required to make a path to other windows
  2b. If a separate register contains the value of a register multiplied by 2
3. As the windows are being applied, they can be performed by multiplying their factors directly instead of multiplying to a separate register.

and square root exponentiations can be determined efficiently by using the windowing method over the entire prime. The exponentations for the square root are slightly different in the two groupings, and the $p = 3 \bmod 4$ general prime is clearly faster as it does not require a square root operation in $\mathbb{F}_p$ in addition to the exponentations.

5 Proposed Technique to Reduce Temporary Registers

In the previous sections, we have not considered the impact of storing intermediate addition chain windows. Here, we propose new techniques that reduce the number of intermediate values needed, while preserving the speed of the addition chains. In software and hardware implementations of inversion, the intermediate storage must be accounted for. This can make a large difference in embedded devices that are limited by the number of values that can be stored. Fast addition chains typically require many more temporary values than something like the binary method, but careful planning can be used to minimize the impact on a register file, for instance. We summarize our observations in Algorithm 6.

5.1 New Techniques

Proposition 4. *Temporary registers can be reduced by creating a short path to reach the first window that involves other windows. The steps along the path that are also windows must be used as registers.*

Proposition 4 leads to a few different techniques to reduce the total number of registers:

Proposition 5. *The windows used in an addition chain do not have to be generated at the start. They only need to be generated in the order that they appear with the windowing method. Thus, after a window is used with no remaining dependencies, its register can be replaced.*

Proposition 5 is simple to see. So long as we can create the first window efficiently, we can recreate the other windows efficiently at a later time. The order of the windows is relevant. In addition, any window that is used twice must be stored as a temporary register so that the cost of generating it is not experienced twice. For example, let us consider we have windows in the order 9, 11, 9, 6, 4, 5. The optimal addition sequence for this toy example is 1, 2, 4, 5, 6, 9, and 11. The shortest addition sequence for the first window is 1, 2, 4, 5, and 9. In reality, we only have to store values for 2, 4, 5, 9, and an accumulator. The other values that are not stored can be recreated from each of these. For instance, $6 = 2 + 4$ and $11 = 2 + 9$. The register holding 9 can also be freed after the second 9 window is applied. There are no more dependencies on it within the window sequence. Likewise, the register holding 2 can be freed after applying the window of 6, and the register holding 4 can be freed after applying the window of 4.

Proposition 6. *New windows can be recreated from pre-existing windows using addition chains at no cost to the complexity of the exponentiation.*

Proposition 6 shows that only the absolutely necessary windows must be stored and that the others can be recreated from multiplications. In the toy example above, we can recreate the windows that are not included by adding the factors to generate the window in sequence. For instance, if we have a window of 11, we would multiply the accumulator by 9 and then multiply it by 2. Alternatively, one could use a temporary register to hold the product of 9 and 2, and then multiply that to the accumulator. In the end, this window costs 2 multiplications to use in the addition chain. Thus, storing the window to a temporary register wastes a register unless the window appears more than once. In our example, 5 and 9 appear twice in the sequence of windows, so a register must hold these values to prevent recreating the window multiple times. There is no reason to store 6 because it is only used once and is not necessary to generate any other windows.

5.2 An Illustrative Example - $2^{253}3^{161}7 - 1$

We demonstrate these techniques with our example in Section 4.2. Originally, this example requires 32 registers since there are 24 windows, 7 intermediate values necessary to complete the addition sequence, and a single register for the accumulator. However, it is worth noting that based on the order of the windows and above propositions, we can reduce the number of registers significantly.

One optimization that we can do is to reach our first window with as few steps as possible. 4057 can be reached in with 21 registers by using the addition chain sequence 1, 2, 3, 4, 7, 8, 16, 23, 27, 29, 37, 38, 40, 77, 115, 123, 125, 165, 205, 243, and an accumulator to perform the other squarings and multiplications up to 4057. The intermediate values must be saved as they are windows that will be used later. Unfortunately, the windows for 43 and 91 occur at the beginning of the exponential sequence and occur multiple times, thus registers must be used to store these as well. 43 requires one step and 91 requires 51, thus 3 additional

Table 3. Comparison of Addition Chains for Square Root Exponentiation by $p = 2^{253}3^{161}7 - 1$

Method	Window Size	$\#\tilde{I}$	$\#\tilde{M}$	$\#\tilde{S}$	Time (μs)	# Registers
Binary	1	0	380	511	2.978	2
K-ary	2	0	224	511	2.435	4
K-ary	4	0	141	511	2.076	16
Standard Window	8	0	87	508	1.877	20
Hybrid Window	8	0	76	508	1.804	21
Window with a Subtraction	8	1	56	508	1.751	21

registers are required. From there, all of the other windows can be reached within a single step. As windows are used in the exponentiation and are not needed to generate other values, these registers can be freed and reused for other windows. Technically, these new windows do not necessarily need to be stored since they can be factored to two of the existing windows, as noted in Proposition 6. Based on data dependency within the window order, 3 additional registers are used. Using these techniques, 5 temporary registers can be saved and 27 registers are required in total.

One more strategy is to free the start of the sequence as their values are used. Indeed, after 47 is obtained, 1, 7, 8, and 16 can each be removed. The rest of the sequence is obtainable. One interesting note is that 1 is not needed since 2 can serve as its window, but after one more squaring. Another interesting use of this technique is that 213 can be applied as a window in two multiplications, even if 8 is not available. One cycle before the window's turn, the factor 4 is multiplied to the accumulator. The accumulator is squared and multiplied with 205, to achieve $213 = 4 \times 2 + 205$. The data dependency technique can also remove 38 and 40 after 243 has been generated since they are not used in any other windows. Thus, we further reduce the register count from 32 at the start to 21, reducing the register usage by 34%.

6 Comparison of Methods

Using the above techniques, we demonstrate the reduced complexity of our method over a standard windowing method in Table 3. We used a Jetson TK1 development board with the GNU Multiprecision (GMP) Library version 6.1.0 to test our addition chain strategies. We used Karatsuba-optimized methods for arithmetic in $\mathbb{F}_{p^2}$ and GMP for arithmetic in $\mathbb{F}_p$. The timing result represents the cost of performing the exponential $p = 2^{253}3^{161}7 - 1$ over $\mathbb{F}_{p^2}$, designed for a square root. The k-ary method is presented in Algorithms 1 and 2. Our hybrid windowing method reduces the total number of multiplications needed by approximately 13% over a standard windowing method. Furthermore, our windowing method with a subtraction reduces the total number of multiplications by 36% at the cost of a fast inversion. The new methods are optimizations of the

Table 4. Comparison of Addition Chains for Smooth Isogeny Primes p_{512}

Exponentiation	$\#I$	$\#M$	$\#S$	Window Addition Sequence Length	#Registers	Max Window Size	c
$p = 2^{253}3^{161}7 - 1$							
$p-2$	0	75	508	31	21	8	13
$\frac{p-1}{2}$	1	56	505	31	21	8	-
$p = 2^{254}3^{158}71 + 1$							
$p-2$	0	79	514	32	19	7	14
$\frac{p-1}{2}$	0	58	507	32	19	7	-
$p = 5^{108}7^{89}732 + 1$							
$p-2$	0	99	505	55	28	10	-
$\frac{p-1}{2}$	0	99	504	55	28	10	-
$p = 5^{108}7^{90}102 + 1$							
$p-2$	0	106	508	54	24	8	-
$\frac{p-1}{2}$	0	106	507	54	24	8	-

windowing strategy, applicable to special isogeny primes of the form $2^a\ell_B^b \cdot f \pm 1$. These optimizations require only a single register over the standard windowing method, but speed up the exponentiation by 3.9% for the hybrid windowing method and 6.8% for the window with a subtraction. Interestingly, the relative ratio of inversion over multiplication in $\mathbb{F}_{p^2}$, $\tilde{I}/\tilde{M}$, was found to be approximately 5 for the Jetson TK1. Thus, the window with a subtraction method reduced the cost of the square root exponentiation by approximately 15 multiplications in $\mathbb{F}_{p^2}$ for 512-bit primes.

We also apply the technique to the three major families with 512 bit primes in Table 4. These results show that the square root exponentiations are faster with the form $p = 2^a\ell_B^b \cdot f + 1$ because a fast inversion is not needed as the least half of the prime is already all '0's. We also compare primes of the form $2^a\ell_B^b \cdot f - 1$ in Table 5. For these two tables, exponentiation by $p - 2$ is for inversion and in $\mathbb{F}_p$ and exponentiation by $\frac{p-1}{2}$ is for the square root and in $\mathbb{F}_{p^2}$. Typically, the total number of registers appeared to be directly related to the max window size and addition sequence to generate the windows. Smaller window sizes required fewer steps to reach the first window and required fewer numbers for the remaining steps. It is also interesting that the optimal max window size did not necessarily scale with the prime size. Generally, windows of size 7-10 appeared the best for our results. These window sizes fit well for these sizes because many of the windows could be generated quickly and there were only additional windows as the max window size got larger. In contrast, the value of c in $2^c - 1$ did appear to scale with the size of the prime. This is to be expected as greater values of c required typically an additional squaring and multiplication, but saved many window multiplications at the end of the prime.

Table 5. Comparison of Addition Chains for Smooth Isogeny Primes of Different Sizes

Exponentiation	$\#I$	$\#M$	$\#S$	Window Addition Sequence Length	#Registers	Max Window Size	c
$p_{512} = 2^{253}3^{161}7 - 1$							
$p-2$	0	76	508	31	21	8	13
$\frac{p-1}{2}$	1	56	506	31	21	8	-
$p_{768} = 2^{379}3^{239}497 - 1$							
$p-2$	0	108	770	49	26	9	16
$\frac{p-1}{2}$	1	84	762	49	27	9	-
$p_{1024} = 2^{509}3^{320}107 - 1$							
$p-2$	0	134	1029	52	28	8	18
$\frac{p-1}{2}$	1	102	1020	52	29	8	-

7 Conclusion

Overall, this paper investigated fast and efficient addition chains for smooth isogeny primes used in the supersingular isogeny Diffie-Hellman scheme. The hybrid windowing method produces fast addition chains for inversion for the $2^a\ell_B^b \cdot f \pm 1$ families by taking advantage of the semi-regular structure of $p-2$. Other primes used in the scheme can use the basic windowing method, but typically require more multiplications. Square root exponentials can also benefit from a fast inversion for $2^a\ell_B^b \cdot f - 1$ or simply from having half of the exponential being zero for $2^a\ell_B^b \cdot f + 1$. The applications of inversions and square roots for isogeny-based cryptography necessitate the need for fast addition chains for fast and secure exponentiations. The hybrid and subtraction windowing methods find addition chains that feature reduced numbers of multiplications and squarings at an insignificant cost to temporary storage, which can be valuable to both the speed and size of ECC over prime curves.

8 Acknowledgment

The authors would like to thank the reviewers for their constructive comments. This material is based upon work supported by the NSF CNS-1464118 and NIST 60NANB16D246 awards.

References

1. Jao, D., De Feo, L.: Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In: Post-Quantum Cryptography–PQCrypto 2011. LNCS 19–34 (2011)
2. Crandall, R., Pomerance, C.: Prime Numbers: A Computational Perspective. Second Edition. Springer (2005)

3. Solinas, J.A.: Generalized Mersenne Numbers. Technical report, University of Waterloo (1999)
4. Itoh, T., Tsujii, S.: A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. Information and Computation 78(3), 171–177 (1988)
5. Koziel, B., Jalali, A., Azarderakhsh, R., Jao, D., Mozaffari-Kermani, M.: NEON-SIDH: Efficient Implementation of Supersingular Isogeny Diffie-Hellman Key Exchange Protocol on ARM. In: 15th International Conference on Cryptology and Network Security, CANS 2016
6. De Feo, L., Jao, D., Plut, J.: Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. Journal of Mathematical Cryptology 8(3), 209–247 (Sep. 2014)
7. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key Compression for Isogeny-Based Cryptosystems. In: Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography. AsiaPKC '16, New York, NY, USA, ACM 1–10 (2016)
8. Costello, C., Longa, P., Naehrig, M.: Efficient Algorithms for Supersingular Isogeny Diffie-Hellman. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Volume 9814 of Lecture Notes in Computer Science. 572–601 (2016)
9. Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M., Jao, D.: Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves. Cryptology ePrint Archive, Report 2016/672 (2016) http://eprint.iacr.org/2016/672.
10. Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M.: Fast Hardware Architectures for Supersingular Isogeny Diffie-Hellman Key Exchange on FPGA. In: Progress in Cryptology – INDOCRYPT 2016, 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings. (2016)
11. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient Compression of SIDH Public Keys. Cryptology ePrint Archive, Report 2016/963 (2016) http://eprint.iacr.org/2016/963.
12. Knuth, D.E.: The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997)
13. Brauer, A.: On addition chains. Bulletin of the American Mathematical Society 45(10), 736–739 (10 1939)
14. Bos, J., Coster, M.: Addition Chain Heuristics. In: Advances in Cryptology — CRYPTO' 89 Proceedings. Springer New York, New York, NY (1990) 400–407
15. Çetin Kaya Koc: Analysis of Sliding Window Techniques for Exponentiation. Computers and Mathematics with Applications 30, 17–24 (1995)
16. Gordon, D.M.: A Survey of Fast Exponentiation Methods. Journal of Algorithms 27(1), 129–146 (April 1998)
17. Moller, B.: Improved Techniques for Fast Exponentiation. In: Information Security and Cryptology — ICISC 2002: 5th International Conference Seoul, Korea, November 28–29, 2002 Revised Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2003) 298–312
18. Adj, G., Rodríguez-Henríquez, F.: Square Root Computation Over Even Extension Fields. Cryptology ePrint Archive, Report 2012/685 (2012) http://eprint.iacr.org/.
19. Silverman, J.H.: The Arithmetic of Elliptic Curves. Volume 106 of GTM. Springer, New York (1992)
20. Vélu, J.: Isogénies entre courbes elliptiques. Comptes Rendus de l'Académie des Sciences Paris Séries A-B 273, A238–A241 (1971)

21. Lagarias, J., Odlyzko, A.: Effective Versions of the Chebotarev Density Theorem. In: Algebraic Number Fields: L-functions and Galois Properties. Symposium Proceedings of the University of Durham 409–464 (1975)
22. Joye, M., Paillier, P., Vaudenay, S.: Efficient Generation of Prime Numbers. In: Cryptographic Hardware and Embedded Systems — CHES 2000: Second International Workshop Worcester, MA, USA, August 17–18, 2000 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2000) 340–354
23. Gueron, S., Krasnov, V.: Fast Prime Field Elliptic-Curve Cryptography with 256-bit Primes. Journal of Cryptographic Engineering 5(2), 141–151 (2014)
24. Montgomery, P. L.: Modular Multiplication without Trial Division. Mathematics of Computation 44(170), 519–521 (1985)
25. Oswald, E., Aigner, M.: Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks. In: Cryptographic Hardware and Embedded Systems — CHES 2001: Third International Workshop Paris, France, May 14–16, 2001 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2001) 39–50

A Appendix

A.1 Addition chains for inversion

Finite-field inversion finds some A^{-1} such that $A \cdot A^{-1} = 1$, where $A, A^{-1} \in \mathbb{F}_p$. This can be computed using Fermat's little theorem, which shows that $A^{-1} = A^{p-2}$. Addition chains can be used to efficiently evaluate these large powers in a constant set of operations, to protect against timing attacks and simple power analysis attacks.

Isogeny-based cryptosystems operate in $\mathbb{F}_{p^2}$, so the inversion in $\mathbb{F}_p$ must be extended as such. We use Equation 2 to perform the inversions in $\mathbb{F}_{p^2}$ with irreducible modulus $x^2 + 1$ (assuming -1 is not a quadratic residue in $\mathbb{F}_p$). We note that an element, a, in $\mathbb{F}_{p^2}$ is represented as a_0 and a_1, where $a_0, a_1 \in \mathbb{F}_p$ and a_1 is the most significant element.

$$a^{-1} = (a_0, a_1)^{-1} = (a_0 \times (a_0^2 + a_1^2)^{-1}, -a_1 \times (a_0^2 + a_1^2)^{-1}) \tag{2}$$

Fast non-constant time inversion. Inversion by Fermat's little theorem is accomplished in constant-time, but it is still slow compared to algorithms such as the Extended Euclidean Algorithm (EEA) and Kaliski's almost inverse. In fact, EEA has a significantly lower time complexity of $O(\log^2 n)$ compared to $O(\log^3 n)$ for Fermat's little theorem. EEA uses a greatest common divisor algorithm to compute the modular inverse of elements a and b with respect to each other, $ax + by = \gcd(a, b)$. We present this alternative for inversion because it makes an inversion term much quicker to compute, which can be used for the square root exponentiations. For our sample implementation, the GMP library incorporates EEA for fast inversion.

A.2 Fast Computation of Square Root

The finite-field square root finds some $A^{1/2}$ such that $A^{1/2} \cdot A^{1/2} = A$, where $A, A^{1/2} \in \mathbb{F}_p$. For the case that $p \equiv 3 \bmod 4$, which is true for primes of the form $2^a 3^b f - 1$, Shank's algorithm can be used to retrieve the square root of the quadratic residue by exponentiating the value by $\frac{p+1}{4}$. However, unlike inversion, not all elements in a prime field have a square root. Thus, there is also a check on the result that if its square and product by the original element is -1, then the square root does not exist.

For the case $p \equiv 1 \bmod 4$, there is also an additional square root operation in $\mathbb{F}_p$. Typically, the method to recover the square root in this case is based on the Tonelli-Shanks algorithm demonstrated in works such as [**?**]. We will not go into the specifics of this square root operation, but the extra overhead for the full square root is significant compared to the case $p \equiv 3 \bmod 4$.

Square roots in $\mathbb{F}_{p^2}$ are trickier than inversion. For the square root in this extension field, we refer to [18], which extends Shank's algorithm for even extension fields. In this work, Algorithms 9 and 10 contain the square root computation over even extension fields when $p \equiv 3 \bmod 4$ when $p \equiv 1 \bmod 4$, respectively.