Title Goes Here

Submitted by

Robert W.V. Gorrie
B.ASc. Computer Science (McMaster University)

Under the guidance of
Douglas Stebila

Submitted in partial fulfillment of
the requirements for the award of the degree of

Masters of Science
in
Computer Science

Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada

Fall Semester 2017

Abstract

¡Abstract here¿

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Recent Research

1.1.1 ¡any sub section here¿

1.1.2 Literature Survey

1.2 Layout of Paper

¡Sub-subsection title¿

some text[1], some more text

¡Sub-subsection title¿

even more text[1], and even more.

1.3 Motivation

[1] ¡footnote here¿

Chapter 2

Technical Background

Over the course of the past decade, Elliptic Curve Cryptography has proven itself a mainstay in the wide world of applied Cryptology. While Isogeny based cryptography does build itself up from the same underlying mathematical concepts as ECC, it simultaneously draws from a deeper, more complicated space of niche geometric notions and algebraic structures.

This chapter will cover the following preliminary topics: isogenies as a mathematical structure, Supersingular Isogeny Diffie Hellman and its related procotols, the Fiat-Shamir construction and its quantum-safe adaptation, and finally isogeny based signature schemes.

2.1 Isogenies

2.1.1 ¡Sub-section title¿

2.1.2 ¡Sub-section title¿

some text[2], some more text

2.1.3 ¡Sub-section title¿

2.1.4 ¡Sub-section title¿

Refer figure 4.1.

2.2 SIDH

2.2.1 Zero-Knowledge Proof of Identity

2.3 Fiat-Shamir Construction

In 1986, A Fiat & Adi Shamir published the paper ”How To Prove YourselfThe Fiat-Shamir Construction (sometimes referred to as the Fiat-Shamir Heuristic)

2.3.1 Unruh's Post-Quantum Adaptation

2.4 Isogeny Based Signatures

Algorithm 1 KeyGen(λ)

Pick a random point S of order $\ell_A^{e_A}$
Compute the isogeny $\phi : E \to E/\langle S \rangle$
pk $\leftarrow (E/\langle S \rangle, \phi(P_B), \phi(Q_B))$
sk $\leftarrow S$
return (pk,sk)

Algorithm 2 Sign(sk, m)

for `i = 1..2`λ **do**
    Pick a random point R of order $\ell_B^{e_B}$
    Compute the isogeny $\psi : E \to E/\langle R \rangle$
    Compute either $\phi' : E/\langle R \rangle \to E/\langle R, S \rangle$ or $\psi' : E/\langle S \rangle \to E/\langle R, S \rangle$
    $(E_1, E_2) \leftarrow (E/\langle R \rangle, E/\langle R, S \rangle)$
    $\mathtt{com}_i \leftarrow (E_1, E_2)$
    $\mathtt{ch}_{i,0} \leftarrow_R \{0, 1\}$
    $(\mathtt{resp}_{i,0}, \mathtt{resp}_{i,1}) \leftarrow ((R, \phi(R)), \psi(S))$
    if $\mathtt{ch}_{i,0} = 1$ **then**
        $\mathtt{swap}(\mathtt{resp}_{i,0}, \mathtt{resp}_{i,1})$
    $h_{i,j} \leftarrow G(\mathtt{resp}_{i,j})$
$J_1 \parallel ... \parallel J_{2\lambda} \leftarrow H(pk, m, (\mathtt{com}_i)_i, (\mathtt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j})$
return $\sigma \leftarrow ((\mathtt{com}_i)_i, (\mathtt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j}, (\mathtt{resp}_{i,J_i})_i)$

Algorithm 3 Verify(pk, m, σ)

$J_1 \parallel ... \parallel J_{2\lambda} \leftarrow H(pk, m, (\mathtt{com}_i)_i, (\mathtt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j})$
for `i = 0..2`λ **do**
    check $h_{i,J_i} = G(\mathtt{resp}_{i,J_i})$
    if $\mathtt{ch}_{i,J_i} = 0$ **then**
        Parse $(R, \phi(R)) \leftarrow \mathtt{resp}_{i,J_i}$
        check $(R, \phi(R))$ have order $\ell_B^{e_B}$
        check R generates the kernel of the isogeny $E \to E_1$
        check $\phi(R)$ generates the kernel of the isogeny $E/\langle S \rangle \to E_2$
    else
        Parse $\psi(S) \leftarrow \mathtt{resp}_{i,J_i}$
        check $\psi(S)$ has order $\ell_A^{e_A}$
        check $\psi(S)$ generates the kernel of the isogeny $E_1 \to E_2$
if all checks succeed **then**
    return 1

If we transcribe the above to the language of the Microsoft SIDH API, we have in essense the following:

Algorithm 4 KeyGen(λ)

(pk, sk) $\leftarrow$ `KeyGeneration_B()`
return (pk,sk)

Algorithm 5 Sign(sk, m)

for `i = 1..2`λ **do**
    $(, R, \psi) \leftarrow$ `KeyGeneration_A(E)`
    $E_1 \leftarrow E/\langle R\rangle$
    $(E_2, E/\langle R, S\rangle) \leftarrow$ `SecretAgreement_B()`
    $(E_1, E_2) \leftarrow (E/\langle R\rangle, E/\langle R, S\rangle)$
    $\texttt{com}[i] \leftarrow (E_1, E_2)$
    $\texttt{ch}[i] \leftarrow_R \{0, 1\}$
    $(\texttt{resp}[i]_0, \texttt{resp}[i]_1) \leftarrow ((R, \phi(R)), \psi(S))$
$J_1 \parallel ... \parallel J_{2\lambda} \leftarrow H(pk, m, (\texttt{com}_i)_i, (\texttt{ch}_i)_i, (h_{i,j})_{i,j})$
return $\sigma \leftarrow ((\texttt{com}_i)_i, (\texttt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j}, ((\texttt{resp})[J_i])$

Chapter 3

Batching Operations for Isogenies

3.1 Batching Procedure in Detail

One of our main contributions is the embedding of a low-level $\mathbb{F}_{p^2}$ procedure into Microsofts pre-existing SIDH library. The procedure in question reduces arbitrarily many unrelated/potentially parallel $\mathbb{F}_{p^2}$ inversions to a sequence of $\mathbb{F}_p$ multiplications & additions, as well as one $\mathbb{F}_p$ inversion.

More specifically, the procedure takes us from n $\mathbb{F}_{p^2}$ inversions to:

- $2n$ $\mathbb{F}_p$ squarings
- n $\mathbb{F}_p$ additions
- 1 $\mathbb{F}_p$ inversion
- $3(n\text{-}1)$ $\mathbb{F}_p$ multiplications
- $2n$ $\mathbb{F}_p$ multiplications

The procedure is as follows:

3.1.1 Projective Space

Because the work of Yoo et al. was built on top of the original Microsoft SIDH library, all underlying field operations (and isogeny arithmetic) are performed in projective space. Doing field arithmitic in projective space allows us to avoid many inversion operations. The downside of this (for our work) is that the number opportunities for implementing the batched inversion algorithm becomes greatly limited.

3.1.2 Remaining Opportunities

There are two functions called in the isogeny signature system that perform a $\mathbb{F}_{p^2}$ inversion: j_inv and inv_4_way. These functions are called once in SecretAgreement and KeyGeneration operations respectively. SecretAgreement and KeyGeneration are in turn called from each signing and verification thread.

Algorithm 6 Batched Partial-Inversion

procedure PARTIAL_BATCHED_INV($\mathbb{F}_{p^2}$[ ] VEC, $\mathbb{F}_{p^2}$[ ] DEST, INT N)
  initialize $\mathbb{F}_p$ $den[n]$
  for `i = 0..(n-1)` **do**
    $den[i] \leftarrow a[i][0]^2 + a[i][1]^2$
  $a[0] \leftarrow den[0]$
  for `i = 1..(n-1)` **do**
    $a[i] \leftarrow$ `a[i-1]*den[i]`
  $a_{inv} \leftarrow$ `inv(a[n-1])`
  for `i = n-1..1` **do**
    $a[i] \leftarrow a_{inv} * dest[i-1]$
    $a_{inv} \leftarrow a_{inv} * den[i]$
  $dest[0] \leftarrow a_{inv}$
  for `i = 0..(n-1)` **do**
    $dest[i][0] \leftarrow a[i] * vec[i][0]$
    $vec[i][1] \leftarrow -1 * vec[i][1]$
    $dest[i][1] \leftarrow a[i] * vec[i][1]$

This means that in the signing procedure there are 2 opportunities for implementing batched partial-inversion with a batch size of 248 elements. In the verify procedure, however, there are 3 opportunities for implementing batched inversion with a batch size of roughly 124 elements.

3.2 Implementation

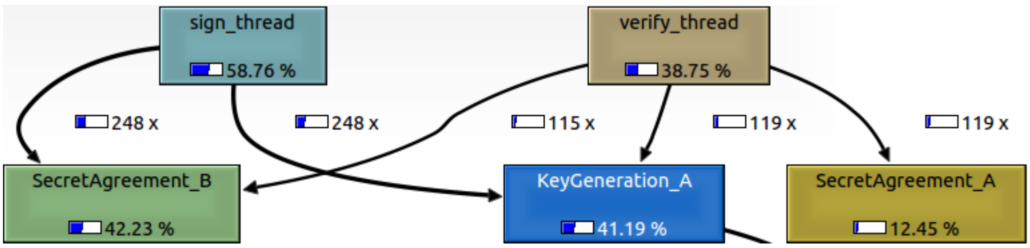

Figure 3.1: ¡Caption here¿

3.3 Results

Two different machines were used for benchmarking. System A denotes a single-core, 1.70 GHz Intel Celeron CPU. System B denotes a quad-core, 3.1 GHz AMD A8-7600.

The two figures below provide benchmarks for KeyGen, Sign, and Verify procedures with both batched partial inversion implemented (in the previously mentioned locations) and not implemented. All benchmarks are averages computed from 100 randomized sample runs. All results are measured in clock cycles.

Procedure	System A Without Batching	System A With Batching
KeyGen	68,881,331	68,881,331
Signature Sign	15,744,477,032	15,565,738,003
Signature Verify	11,183,112,648	10,800,158,871

Procedure	System B Without Batching	System B With Batching
KeyGen	84,499,270	84,499,270
Signature Sign	10,227,466,210	10,134,441,024
Signature Verify	7,268,804,442	7,106,663,106

System A: With inversion batching turned on we notice a 1.1 % performance increase for key signing and a 3.5 % performance increase for key verification.

System B: With inversion batching turned on we a observe a 0.9 % performance increase for key signing and a 2.3 % performance increase for key verification.

3.3.1 Analysis

It should first be noted that, because our benchmarks are measured in terms of clock cycles, the difference between our two system clock speeds should be essentially ineffective.

In the following table, "Batched Inversion" signifies running the batched partial-inversion procedure on 248 $\mathbb{F}_{p^2}$ elements. The procedure uses the binary GCD $\mathbb{F}_p$ inversion function which, unlike regular $\mathbb{F}_{p^2}$ montgomery inversion, is not constant time.

Procedure	Performance
Batched Inversion	1721718
$\mathbb{F}_{p^2}$ Montgomery Inversion	874178

Do performance increases observed make sense?

Chapter 4

Compressed Signatures

4.1 Compression of Public Keys

We discussed rejection sampling A values from signature public keys until we found an A that was also the x-coord of a point. After some simple analysis, however, we found that it was extremely unlikely for A to be a point on the curve.

4.1.1 ¡Sub-section title¿

4.1.2 ¡Sub-section title¿

some text[2], some more text

4.1.3 ¡Sub-section title¿

4.1.4 ¡Sub-section title¿

Refer figure 4.1.

Figure 4.1: ¡Caption here¿

4.1.5 ¡Sub-section title¿

4.2 Implementation

4.3 Results

Chapter 5

Discussion & Conclusion

5.1 Results & Comparisons

5.2 Additional Opportunities for Batching

5.3 Future Work

¡Conclusion here¿

Acknowledgments

¡Acknowledgements here¿

¡Name here¿

¡Month and Year here¿
National Institute of Technology Calicut

References

[1] ¡Name of the reference here¿, <urlhere>

[2] ¡Name of the reference here¿, <urlhere>