2-2021

Generation, Verification, and Attacks on Elliptic Curves and their Applications in Signal Protocol

Tanay Pramod Dusane
tpd4203@rit.edu

Generation, Verification, and Attacks on Elliptic Curves and their Applications in Signal Protocol

by

Tanay Pramod Dusane

THESIS

Presented to the Faculty of the Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology

in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Computer Science

Rochester Institute of Technology
February 2021

Generation, Verification, and Attacks on Elliptic Curves and their Applications in Signal Protocol

Approved by

Supervising committee:

Stanisław Radziszowski, Chair

Monika Polak, Reader

Anurag Agarwal, Observer

Abstract

Elliptic curves (EC) are widely studied due to their mathematical and cryptographic properties. Cryptographers have used the properties of EC to construct elliptic curve cryptosystems (ECC). ECC are based on the assumption of hardness of special instances of the discrete logarithm problem in EC. One of the strong merits of ECC is providing the same cryptographic strength with smaller key size compared to other public key cryptosystems. A 256 bit ECC can provide similar cryptographic strength as 3072 bit RSA cryptosystem. Due to smaller key sizes, elliptic curves are an attractive option in devices with limited storage capacity. It is therefore essential to understand how to generate these curves, verify their correctness and assure that they are resistant against attacks.

The security of an EC cryptosystem is determined by the choice of the curve that is used in that cryptosystem. Over the years, a number of elliptic curves were introduced for cryptographic use. Elliptic curves such as FRP256V1, NIST P-256, Secp256k1 or SM2 curve are widely used in many applications like cryptocurrencies, transport layer protocol and Internet messaging applications. Another type of popular curves are Curve25519 introduced by Dan Bernstein and Curve448 introduced by Mike Hamburg, which are used in an end to end encryption protocol called `Signal`. This protocol is used in popular messaging applications like `WhatsApp`, `Signal Messenger` and `Facebook Messenger`.

Recently, there has been a growing distrust among security researchers against the previously standardized curves. We have seen backdoors in the elliptic curve cryptosystems like the `DUAL_EC_DRBG` function that was standardized by NIST, and suspicious "random seeds" that were used in NIST P-curves. We can say that many of the previously standardized curves lack transparency in their generation and verification. In this thesis, we focus on transparent generation and verification of elliptic curves. We generate curves based on NIST standards and

propose new standards to generate special type of elliptic curves. We test their resistance against the known attacks that target the ECC. Finally, we demonstrate ECDLP attacks on small curves with weak structure.

Acknowledgements

A lot of people have been with me throughout my incredible journey at Rochester Institute of Technology. I would like to thank all the faculty members, friends and my advisors for supporting me throughout this process.

I would first like to thank my thesis advisor Professor Stanisław Radziszowski for believing in me and my abilities to work on this topic. His guidance has been invaluable throughout the cryptography course, independent study and my master's thesis. He always kept his door and zoom meetings open to help me with any roadblocks I have faced during this process. I have been incredibly fortunate to be able to work under his guidance.

I would also like to thank my thesis reader Professor Monika Polak for providing her invaluable feedback. Thanks to Professor Anurag Agarwal for his feedback and being a part of my committee as observer.

Special thanks to me Graduate Advisor Rebecca O'Connor for being a source of encouragement and help during this journey. Without her, this chapter would be incomplete.

I would also like to thank my parents Pramod Dusane and Anjusha Dusane for believing in me and letting me pursue my dreams. Without their sacrifices and hard work, this journey would've been impossible. Thanks to my sister Chinmayee Dusane for being a source of support and hope. And finally my friends Aniruddha, Rishabh and Alok for being supportive throughout this journey.

Contents

List of Symbols

$\#E$	Total number of points on the curve E
$\mathbb{F}_p$	Finite Field with p elements
E	Elliptic curve
$E_{E,a,d}$	Edwards curve with a, d parameters
$E_{K,a}$	Koblitz curve with a parameter
$E_{M,A,B}$	Montgomery curve with A, B parameters
G	Generator of the curve
$GF(2^m)$	Galois field with 2^m elements
h	Cofactor of the curve
n	Order of the generator G
tr	Trace of elliptic curve
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
X3DH	Extended Triple Diffie-Hellman
XEdDSA	Extended Edwards Curve Digital signature algorithm

List of Tables

List of Figures

Part I

Elliptic curves theory and algorithms

1 Introduction and thesis overview

Elliptic curves are defined by certain cubic equations with two variables. The simplified Weierstrass equation is often used for cryptography purpose. It is defined as,

$$y^2 = x^3 + ax + b,$$

where a, b are real numbers and x and y take on the values in the real numbers. EC that are defined over finite fields are of stronger importance in cryptography. Typically the curves used for cryptographic purpose are defined over a large prime field $\mathbb{F}_p$ or Galois field $GF(2^n)$, of around 2^{250} elements. A wide range of standardized elliptic curves are deployed over the Internet. Standards like ANSSI, NIST, OSCCA, Brainpool set out a list of requirements that the curves should satisfy in order to be a standardized curve. Recently there has been some concerns about the security of the previously standardized curves. Curves like FRP256V1 standardized by ANSSI and SM2 curve standardized by OSCCA are distributed without any public justification. That is, their parameters were chosen without any explanation for how they were chosen [4]. This should raise suspicions about any potential backdoors that could be present in these curves.
The Snowden leak in 2013 showed that NSA deliberately inserted a backdoor in a CSPRNG function called `DUAL_EC_DRBG` used in many ECC. NIST standardized the dual elliptic curve deterministic random bit generator function

despite security researchers deeming it to be insecure [21].
This increased a level of distrust among cryptographers for previously standardized curves. Bernstein et al. demonstrate the generation of vulnerable curves that follow all the standardization requirements. One could assume that the standardization agencies distribute curves that can lead the users to believe the curves are secure but the agencies are able to break them [4].
It is therefore essential to understand the criteria for generating the curves and verifying the parameters of an EC. The security of an ECC is based on the assumption of hardness of special instances of the discrete logarithm problem in EC. This hardness depends primarily on the parameters of an EC and the choice of the starting point. Flori et al. discuss the necessary criteria that a curve must satisfy in order to be deemed secure [11]. We cover these criteria in more detail in the thesis.
The purpose of this thesis is to explore the generation and verification process of EC. We develop software that generates curve parameters adhering to all the necessary requirements. Finally, we test the security of the generated curves using a tool developed by Dan Bernstein and Tanja Lange. The end goal is to generate curves that can be used as alternative curves in well known end to end encryption protocols like `Signal`.

1.1 Thesis overview

There has been growing distrust for previously standardized curves due their weaknesses or unexplained generation processes. With the ever increasing usage of end-to-end encrypted protocols, it is essential that the cryptographic primitives used in these protocols are publicly trusted. Dan Bernstein and his colleagues paved the way for programmers to move from insecure standardized elliptic curves to new family of secure and trusted elliptic curves [5]. Curve25519 along with its implementation in `Signal` protocol popularized the end-to-end encryption methodology. In this thesis, we study the properties of elliptic curve that are used or could be used in `Signal` protocol. We study

elliptic curve structure and analyze its security. We focus on the criteria that the curves have to satisfy in order to be "safe". We analyze security criteria recommended by various cryptographers. Using the `SafeCurves` criteria, we propose Montgomery elliptic curves that are secure to use in protocols such as `Signal`. Finally we present the security analysis of our curves and demonstrate their usage in small elliptic curve cryptosystem.

1.2 Thesis goals

G1: Overview the use of EC in cryptosystems. To generate an elliptic curve, it is important to understand the underlying concepts of elliptic curves and its group operations.

G2: Analyze security of EC. We focus on requirements that make EC secure.

G3: Construct software to generate secure EC.

G4: Verify the correctness of generated curves with respect to the guidelines. Adhering to the requirements, we generate curve parameters and check if the curve passes all the necessary requirements.

G5: Test the usability of the generated curve in ECC. Once the generated curve passes all the necessary requirements, we will construct toy cryptosystem using the generated curve.

G6: The curves we generate should be resistant against the known ECDLP attacks. However, we will demonstrate ECDLP attacks on small EC.

1.3 Outcomes

G1: We studied the use of EC in different cryptosystems. We observed the different families of EC and their properties and features. These features helped us understand why specific curves are chosen in certain cryptosystems.

G2: We analyzed the security of EC by studying the different types of attacks that can affect the security of ECC.

G3: We constructed a software that would perform the EC arithmetic and EC conversions on three different families of curves. We use GMP library to efficiently and accurately compute ECC arithmetic operations.

G4: We verified the security of generated curves with respect to the `SafeCurves` criteria. We generated the curve parameters that passed all the necessary security requirements.

G5: We constructed a toy EC Cryptosystem that is based on the generated the curves that we have generated. We demonstrate the use of our curve in different elliptic curve cryptosystems.

G6: We implemented algorithms to solve ECDL problem on small EC. We verified that the resistance of our generated curves against modern ECDLP solvers.

2 Elliptic curves

Elliptic curves are the basis of elliptic curve cryptosystems. Elliptic curves are defined by certain cubic equations with two variables. These curves can be defined over real number, modulo a prime number, or over a finite field $\mathbb{F}_p$. For cryptographic use, we focus on elliptic curves that are defined over finite fields $\mathbb{F}_p$.

Definition 1 *Consider a finite field $\mathbb{F}_p$. Let $a, b \in \mathbb{F}_p$ be constants such that $4a^3 + 27b^2 \neq 0$. A nonsingular elliptic curve is the set E of solutions $(x, y) \in \mathbb{F}_p^2$ to the equation*

$$y^2 = x^3 + ax + b,$$

together with a special point $\mathbb{O}$ called the point at infinity. This equation is called a Weierstrass equation. We will cover other type of equations in later sections.

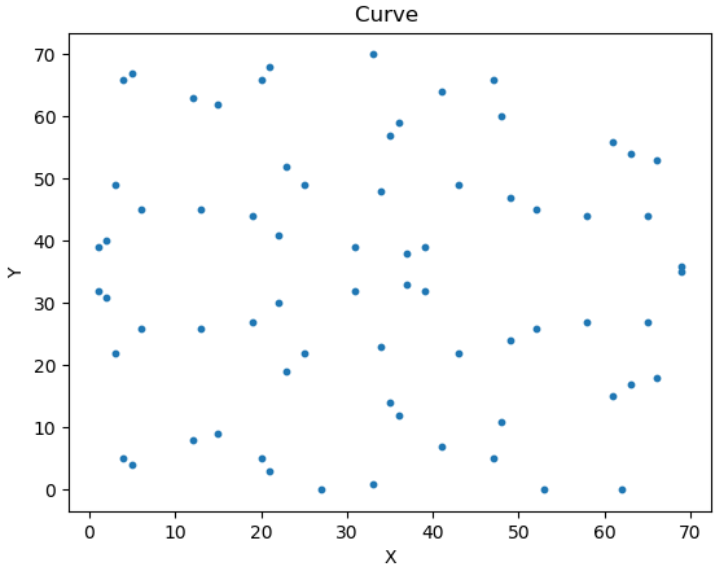


Figure 1: Plot of $y^2 = x^3 + x + 28$ over $\mathbb{Z}_{71}$

2.1 Group operations on elliptic curves

Group operations such as addition and scalar multiplication are performed using the points on the curve. There are two cases of group operations that we consider: Addition of two distinct points on the curve and addition of a point on the curve with itself (point doubling).

2.1.1 Point addition

Point addition requires "adding" two distinct points on the elliptic curve to produce a third point. Since we are producing a third point by "adding" two points, the operation is arbitrarily named as "addition" [19].

Consider two points $P, Q \in E$ where $P(x_1, y_1)$ and $Q(x_2, y_2)$, we define group operation addition as $P(x_1, y_1) + Q(x_2, y_2) = R(x_3, y_3)$. The algebraic formula to compute R is as follows:

Case 1: $x_1 \neq x_2$

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \\ where\ \lambda &= \frac{y_2 - y_1}{x_2 - x_1}. \end{aligned}$$

Case 2: $x_1 = x_2, y_1 = -y_2$

$$(x_1, y_1) + (x_1, -y_1) = \mathbb{O}.$$

Case 3: $x_1 = x_2$ and $y_1 = y_2$

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \\ where\ \lambda &= \frac{3x_1^2 + a}{2y_1}. \end{aligned}$$

2.1.2 Point scalar multiplication

Point scalar multiplication is an operation where a point P is added to itself k number of times, where $3 \leq k \leq n$ and n is the order of the base point of curve E. Point multiplication operation on EC corresponds to exponentiation [19]. It is the basis of elliptic curve discrete log problem (ECDLP).

$$\underbrace{P + P + P + P + ... + P}_{k\ times} = kP.$$

There are various methods to efficiently compute point multiplication. Such as double and add algorithm, Montgomery ladder and sliding window approach.

2.2 Elliptic Curve Discrete Logarithm Problem (ECDLP)

Elliptic curve discrete logarithm problem (ECDLP) is an analog of the discrete log problem (DLP). Consider a base point P and another point Q such that $P, Q \in E$, the problem asks to find integer d, where $1 \leq d \leq \#E$, such that $P = dQ$. Here, the total number of points on E is denoted by $\#E$.
Security of ECC is determined by the hardness of ECDLP problem. Size of the curve and properly chosen base point can make the ECDLP problem hard. For an EC to be secure, computing the value of d should be an infeasible task.

2.3 Elliptic curves in cryptography

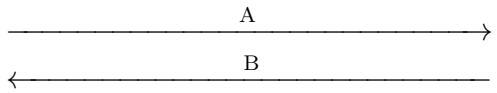
Elliptic curves are widely used in the modern cryptography due to their properties. Unlike RSA, the EC key size d is smaller but it provides the same level of security. This makes the usage of EC very compelling in computers that have limited storage and computational capacity. For this reason, EC is used for digital signature algorithms, key exchange algorithms, key agreement algorithms and pseudo-random generators. EC is a fundamental building block in end-to-end encryption protocol `Signal` and for cyrptocurrencies like `Bitcoin` and `Ethereum`.

2.3.1 Elliptic Curve Diffie Hellman (ECDH)

Diffie Hellman key exchange (DHKE) protocol is used to securely exchange keys between two parties over an insecure channel. Elliptic curve Diffie Hellman (ECDH) is a variation of DHKE which uses EC. DHKE uses modular arithmetic to compute the keys whereas ECDH uses elliptic curve arithmetic. Before initiating the ECDH protocol, both parties have to agree upon an elliptic curve and its primitives such as base point. Once the primitives are agreed upon, the protocol works as follows:

Alice Bob

choose $k_{prA} = a \in \{2, 3, ..., \#E - 1\}$ choose $k_{prB} = b \in \{2, 3, ..., \#E - 1\}$
compute $k_{pubA} = aP = A = (x_A, y_A)$ compute $k_{pubB} = bP = B = (x_B, y_B)$



compute $aB = T_{AB}$ compute $bA = T_{AB}$

Shared secret between Alice and Bob is $T_{AB} = (x_{AB}, y_{AB})$

2.3.2 Elliptic Curve Digital Signature Algorithm

Digital Signal Algorithm(DSA) is used to sign and verify messages. The messages are signed using the private key of the signer and verified using the public key of the signer. DSA key generation operations are defined under $\mathbb{Z}_{|}^{*}$. Unlike DSA, ECDSA is defined under a group of an elliptic curve.

- **Key Generation of ECDSA:**
  - Choose a large prime p and an elliptic curve E defined over Z_p
  - Choose a point A on E such that it generates a cyclic group of prime order n.
  - Let $P = \{0, 1\}, A = Z_p^* \times \mathrm{Z}_p^*$, and define $K = f(p, n, E, A, m, B) : B = mA$ where $0 \leq m \leq n - 1$.
  - Public key is (p, n, E, A, B), and m is the private key
- **Signature generation:**
  - Choose a random ephemeral key $k, 1 \leq k \leq n - 1$
  - $sig_K(x, k) = (r, s)$ where,

* $kA = (u, v)$
* $r = u \mod n$
* $s = k^{-1}(\text{SHA512(x)} + mr) \mod n$

- **Signature verification:**
  - Compute $w = s^{-1} \mod n$
  - $i = w \times \text{SHA512(x)} \mod n$
  - $j = wr \mod n$
  - $(u, v) = iA + jB$
  - $\mathbf{ver}_K(x, (r, s)) = true \Leftrightarrow u \mod n = r$

2.3.3 `Signal` protocol

`Signal` protocol uses extended triple Diffie Hellman (X3DH) key agreement protocol to establish a shared secret key between the two communicating parties. And it uses extended Edwards curve digital signature algorithm (XEdDSA) for verification of messages and public keys used in communication. Here, we focus on X3DH as it uses XEdDSA in one of its steps to establish shared secret key. X3DH provides forward secrecy and cryptographic deniability. We focus on asynchronous communication scenario between two parties "Alice" and "Bob" , where Bob is offline and Alice wants to send a message to Bob.
Before setting up X3DH protocol in an application, We must decide upon following parameters.

Name	Definition
curve	Curve25519 or Curve448
hash	A 256 or 512-bit hash function (e.g. SHA-256 or SHA-512)
info	Information about the protocol (e.g `My Application`)

Cryptographic Notations:

- $X||Y$: Concatenation of byte sequences X and Y.
- $DH(PK1, PK2)$: Shared secret obtained from Elliptic Curve Diffie Hellman function using public keys *PK1* and *PK2*.
- $Sig(PK, M)$: Byte signature generated by XEdDSA function with M as message and PK as public key.
- $KDF(KM)$: Key Derivation function that derives a secret key that is used to encrypt the message.

Keys:

- IK_A - Alice's Identity Key.
- EK_A - Alice's Ephemeral Key.
- IK_B - Bob's Identity Key.
- EK_B - Bob's Ephemeral Key.
- SPK_B - Bob's signed prekey.
- OPK_B - Bob's one time prekey.

The keys used in X3DH are elliptic curve public keys. All of the keys have their corresponding private keys. Identity keys are long term keys. Ephemeral keys are generated on every iteration of the protocol. Prekeys are uploaded to the server before initiation of any communication. Each party signs their prekeys and uploads it to the server. Prekeys ensure forward secrecy.

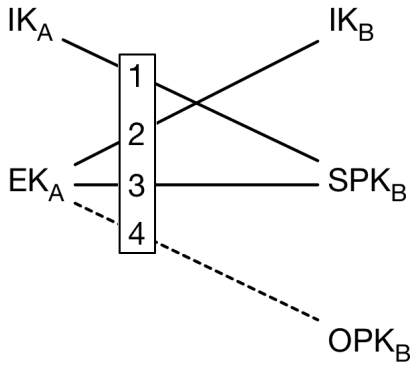
Sending initial message:

Each communicating party publishes their identity keys (IK), signed prekeys (SPK), prekey signatures $Sig(IK, Encode(SPK))$ and a bunch of one time prekeys (OPK_1,OPK_3,OPK_3,...). Identity keys are uploaded once while signed prekeys and its signatures are uploaded in intervals (e.g every week or every month).
To perform the X3DH Key Agreement, Alice has to fetch a set of keys of Bob from server. These keys are :

- Identity key of Bob IK_B
- Signed prekey of Bob SPK_B
- Prekey signature of Identity key and signed private key $Sig(IK_B, Encode(SPK_B))$
- Optional Bob's one time prekey OPK_B

The one time prekey is optional. If it exists then the server should fetch it to Alice and then delete it. Alice generates her own set of ephemeral keys EK_A Alice then generates a shared secret key that is used to encrypt messages between her and Bob. This shared secret key is calculated by performing three or four Diffie Hellman Key exchanges depending upon the presence of Bob's one time prekey.

$DH1 = DH(IK_A, SPK_B)$
$DH2 = DH(EK_A, IK_B)$
$DH3 = DH(EK_A, SPK_B)$
If OPK_B exists,
$DH4 = DH(EK_A, OPK_B)$



The above Diffie Hellman exchanges provide two properties-

- $DH1$ and $DH2$: Mutual Authentication
- $DH3$ and $DH4$: Forward Secrecy

Secret Key is calculated after all the exchanges are performed.
$SK = KDF(DH1||DH2||DH3||DH4)$
This secret key is used for encrypting the messages between Alice and Bob.

2.4 Domain parameters

Generator of the curve:
The generator G of the curve is a special point on the curve that can generate the entire group when repeatedly added to itself. It is also referred as the base point.

Order of the curve:
The order of the curve $\#E$ is the total number of points on the EC.

Order of the base point:
The order of base point G is the smallest positive integer n such that $nG = \mathbb{O}$.

Cofactor of the curve:
The cofactor h denotes the number of subgroups that are generated by the generator G and order n. We have the relation $n \cdot h = \#E$. For ECDLP security, large prime order n and small cofactor h is desired.

2.5 Special type of curves

This section covers additional types of elliptic curves that are used in ECC.

2.5.1 Edwards Curves

Edwards curves are a family of elliptic curves that were first introduced by Harold Edwards, in 2007. The original curve discussed by Edwards was a normal form of elliptic curve with the equation [9]

$$x^2 + y^2 = a^2 + a^2x^2y^2.$$

Bernstein and Lange presented formulas for fast addition and point doubling of the coordinates on the Edwards curve. They generalized the addition law to the curves $x^2 + y^2 = a^2 + a^2x^2y^2$ which covers more elliptic curves over a finite field than $x^2 + y^2 = a^2 + a^2x^2y^2$. These curves are isomorphic to the curve [3]

$$x^2 + y^2 = 1 + dx^2y^2. \tag{1}$$

We refer to such curves as Edwards Curves.

2.5.2 Twisted Edwards Curve

An Edwards curve is a twisted Edwards curve with $a = 1$ [3].

Definition 2 *: Fix a field K with $char(K) \neq 2$. Fix distinct nonzero elements $a, d \in K$. The twisted Edwards curve with coefficients a and d is the curve*

$$E_{E,a,d} : ax^2 + y^2 = 1 + dx^2y^2.$$

Addition of two points $P(x_1, y_1)$ and $Q(x_2, y_2)$ on twisted Edwards curve is given as:

$$P \oplus Q = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2} \right),$$

Point doubling on twisted Edwards curve $P = Q = (x_1, y_1)$ is given as:

$$P \oplus P = \left(\frac{2x_1y_1}{ax_1^2 + y_1^2}, \frac{y_1^2 - ax_1^2}{2 - ax_1^2 - y_1^2} \right).$$

2.5.3 Curve 448

Curve 448 is an Edwards curve designed by Mike Hamburg. It provides up to 224 bits of security [14]. Curve 448 is one of the curve that can be used in `Signal` protocol and `WhatsApp` Messenger. The parameters of the curve 448 defined as per the RFC 7748 [18] are:

p	$2^{448} - 2^{224} - 1$
d	-39081
n	2^{446} - 0x8335dc163bb124b65129c96fde933d8d723a70aadc873d6d5 4a7bb0d
h	4
$X(G)$	224580040295924300187604334099896036246789641632564134246125461686950415467406032909029286935795328257803207514644617367460263524771 0
$Y(G)$	298819210078481492676017930443930673437544040154080242095928241372331506189835876003537865541878473398230323350346250053154506283266 0

$X(G)$ and $Y(G)$ are the (x, y) coordinates of the base point G.

2.5.4 Montgomery Curves

Montgomery curve is another form of elliptic curve that was introduced by Peter Montgomery in 1987. A Montgomery curve defined over field K is equivalent to a twisted Edwards curve over a field K [3].

Definition 3 *: Fix a field K with $char(K) \neq 2$. Let $A \in K\backslash\{-2, 2\}$ and $B \in K\backslash\{0\}$. Montgomery curve is defined as*

$$E_{M,A,B} : Bv^2 = u^2 + Au^2 + u.$$

This equivalence between both the curves is called birational equivalence because there exists a mapping $\varphi : E_1 \rightarrow E_2$ and an inverse mapping

$\varphi^{-1} : E_2 \to E_1$ between the points on twisted Edwards curve and Montgomery curve. Given a point on any one curve, we can obtain an equivalent point on Montgomery curve using the mapping [3].

The birational maps between the curves:

- A map between $E_{E,a,d}$ and $E_{M,A,B}$ where $A = \dfrac{2(a+d)}{a-d}$ and $B = \dfrac{4}{a-d}$,

$$(x, y) \to (u, v) = \left(\frac{1+y}{1-y}, \frac{1+y}{(1-y)x}\right),$$

- A map between $E_{M,A,B}$ and $E_{E,a,d}$ where $a = \frac{A+2}{B}$ and $d = \dfrac{A-2}{B}$,

$$(u, v) \to (x, y) = \left(\frac{u}{v}, \frac{u-1}{u+1}\right).$$

2.5.5 Curve 25519

Curve 25519 is a popular curve that was introduced by Dan Bernstein in 2005. Curve 25519 is widely used in `Signal` protocol, `WhatsApp` Messenger, Secure Shell, Transport Layer Security and cryptocurrencies like ZCash.

The parameters of the curve 25519 defined as per the RFC 7748 [18] are:

p	$2^{255} - 19$
A	486662
n	2^{252} + 0x14def9dea2f79cd65812631a5cf5d3ed
h	8
$U(G)$	9
$V(G)$	14781619447589544791020593568409986887264606134616475289648818377555862374 01

$U(G)$ and $V(G)$ are the (u, v) coordinates of the base point G.

2.5.6 Koblitz Curves

Koblitz curves are defined over binary fields GF(2^k). They were introduced by Neal Koblitz.

Definition 4 *: Fix a binary field K. Let $a \in K\backslash\{0, 1\}$ and $B \in K\backslash\{1\}$. Koblitz curve is defined as*

$$E_{K,a} : y^2 + xy = x^3 + ax^2 + 1.$$

Koblitz are not adopted as widely as other families of curves. Due to security problems, they may be deprecated in the future.

2.6 Verification of birational maps between Curve25519 and Ed25519

The equation of the Curve25519 is given as

$$v^2 = u^2 + 486662u^2 + u \bmod 2^{255} - 19.$$

and the coordinates of the generator G are (G_u, G_v). Now, we derive an edwards curve birationally equivalent to Curve25519 using the maps discussed in 2.5.4. The parameters we get are:

$$a = \frac{A+2}{B} = \frac{486662+2}{1} = 486664 \text{ and,}$$
$$d = \frac{A-2}{B} = \frac{486662-2}{1} = 486660.$$

The curve that is birationally equivalent to the Curve25519 is

$$486664x^2 + y^2 = 1 + 486660x^2y^2 \bmod 2^{255} - 19.$$

We can also map the generator as,

$$(G_x, G_y) = \left(\frac{G_u}{G_v}, \frac{G_{u-1}}{G_{v+1}} \right)$$

The value of generator can be verified by substituting it in the formula above. The Edwards curve Ed25519 defined in RFC 8032 has a different equation than derived above. This is due to a scaling and substitution factor that is out of scope for this thesis.

2.7 Contribution to thesis goals

This section contributes to goal G1. To generate an elliptic curve, it is important to understand the underlying concepts of elliptic curves and its group operations. In this section we cover the background required for studying the generation of elliptic curves.

3 Counting Points on Elliptic Curves

An important problem to study while constructing ECs is counting points on an EC. The total number of points on an EC is denoted by $\#E$. Counting points is essential for determining the order of the curve, which is not a trivial task. The total number of points on the EC is also called as order of an EC. For a curve to be secure, the order of the curve must be a large prime p or a large prime p times a very small cofactor h.
A cofactor $h = 1$ gives optimal security and having a small cofactor can provide performance enhancements [11]. In order to make the subgroup attack infeasible, the order of the curve should have a large prime factor. Pohlig-Hellman algorithm can be used to attack the group of order n, but the attack becomes hard if the order of subgroup is a large prime. Stinson suggests that, for a curve to be resistant against ECDLP attacks, the order of subgroup of the curve should be $\geq 2^{224}$ [23].

To construct ECC, we have to calculate the order of the curve. Finding the order of the curve however is not a trivial job. Hasse's theorem gives us an approximation of $\#E$.

Hasse's Theorem: *Given an elliptic curve E over Z_p, the number of points on the curve denoted by $\#E$ is bounded by [19]*

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}.$$

One way to compute the order of the curve by a naive approach, that is by substituting all the elements of F_p in the curve equation. However, that would be an exhaustive task for a large p. Luckily, algorithms such as Schoof's and SEA can compute $\#E$ in polynomial time.

3.1 Point counting algorithms

Schoof's Algorithm:
Schoof's algorithm is used to count the points on the curve. It is very efficient when the field of the curve is larger than 2^{160}. The complexity of this algorithm is $\mathcal{O}(ln^8 p)$. To understand this algorithm, lot of concepts of advanced number theory are required which we will study during the thesis work. The idea behind Schoof's algorithm is to compute trace of Frobenius $t \mod l$ such that l is a small prime number in K and then use Chinese remainder theorem to compute t. Then we can compute $\#E = p + 1 - t$ [1]. Sundriyal suggests improvement in Schoof's algorithm by using Shank's BSGS algorithm within the internal searching of l. This can further increase the speed of the algorithm [24].

Schoof-Elkies-Atkins (SEA) algorithm:
SEA algorithm is an improvement over Schoof's algorithm. The drawback of Schoof's algorithm is with the degree of division polynomials. SEA algorithm deals with this drawback of Schoof's algorithm and presents an improvement. It reduces the complexity of the algorithm from $\mathcal{O}(ln^8 p)$ to $\mathcal{O}(ln^5 p)$. This algorithm can count the order of curve over $\mathbb{F}_p$ with p over 500 digits [24].

3.2 Experimental work

We calculate the order of the curves using `Schoof's`algorithm. We used a compiled binary of `Schoof's` algorithm, authored by Michael Scott which is found in `MIRACL` library. The purpose of the experiment was to test the `Schoof's` algorithm software's correctness and efficiency. To cross reference the efficiency, we compute the order of same curves with `SageMath`. `SageMath` uses `Schoof-Elkies-Atkin` (SEA) algorithm which is an improved over `Schoof's` algorithm to compute the order of the curves. The detailed

Algorithm: Schoof's Algorithm
Let p be a prime such that $p > 3$. For a curve $E_{a,b}$ this algorithm returns the value of $t(\text{mod } l)$, where l is in a set of much smaller primes than p and the curve order is $\#E = p + 1 - t$. For more details refer to [7] and [3].

Input: a, b, p
Output: $\#E_{a,b}$

1. [For loop]
   For each l in the set of small primes, repeat steps 2 and 3.

2. [Check $l = 2$]
   if $(l == 2)$ {
      $g(X) = \gcd(X^p - X, X^3 + aX + b)$; //Polynomial gcd in $F_p[X]$.
      if $(g(X) == 1)$ return 0; //$T \equiv 0 \pmod 2$, so order $\#E$ is even.
      return 1; $\#E$ is odd.
   }

3. [Analyze relation (7.10)]
   $\overline{p} = p \bmod l$;
   $u(X) = X^p \bmod (\Psi_{l,p})$ //$\Psi_{l,p} = \Psi_l \pmod p$
   //That is, $v(X) = Y^{p-1} \bmod (\Psi_{l,p})$.
   //$P_0 = (X^p, Y^p)$.
   $P_0 = (u(X), Yv(X))$
   $P_1 = (u(X)^p \bmod (\Psi_l, p), Yv(X)^{p+1} \bmod (\Psi_l, p))$
   //$P_0 = (X^{p^2}, Y^{p^2})$.
   Cast $P_2 = [\overline{p}](X, Y)$ in rational form.
   if $(P_1 + P_2 == O)$ return 0; //$\#E = p + 1 - t$ *with* $t \equiv 0 \pmod l$
   $P_3 = P_0$
   for $(1 \le k \le l/2)$ {
      if (X-coordinates of $(P_1 + P_2)$ and P_3 match) {
         if (Y-coordinates also match) return k;
         return $l - k$;
      }
      $P_3 = P_3 + P_0$;
   }

4. Find out the value of t for all the l and find the exact value of t using the Chinese remainder theorem. The actual order can be calculated and confirmed with the help of Hasse's bound.

Figure 2: Schoof's algorithm taken from [24]

output of our experiments can be found in appendix 1.

Schoof's algorithm can count the points on a Weierstrass curve. `SEA` algorithm in `SageMath` can count points on the Montgomery and Weierstrass

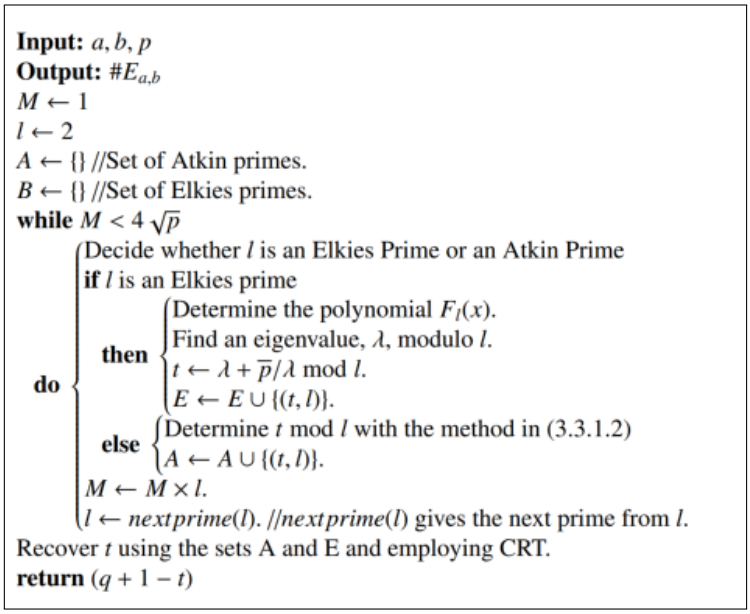


Figure 3: Schoof-Elkies-Atkins (SEA) algorithm taken from [24]

curves. Hence, using birational mappings, we use `Wei2551` which is a Weierstrass form of Curve25519. Similarly, for Edwards curves, we use birational mappings to convert them to appropriate Montgomery or Weierstrass form in order to compute order of the curve.

3.2.1 Curve25519

Wei25519: $y^2 = x^3 + a * x + b \bmod p$

a	19298681539552699237261830834781317975544997444273427339909597334573241639236
b	55751746669818908907645289078257140818241103727901012315294400837956729358436
p	$2^{255} - 19$

Using `Schoof's` algorithm:

NP=57896044618658097711785492504343953926856930875039260848015607
506283634007912

Using `SEA` algorithm:

```
sage: ec.order()
NP = 57896044618658097711785492504343953926856930875039260848015
607506283634007912
```

3.2.2 NIST P-256

NIST P-256: $y^2 = x^3 - 3x + b \bmod p$

a	-3
b	1
p	41058363725152142129326129780047268409114441015993725554835256314039467401291

Using `Schoof's` algorithm:
NP=11579208921035624876269744694940757352999695522413576034242225
9061068512044369

Using `SEA` algorithm:

```
sage: ec.order()
NP = 11579208921035624876269744694940757352999695522413576034242225
259061068512044369
```

3.2.3 Curve448

Curve448: $By^2 = x^3 + A * x^2 + x \bmod p$

A	156326
B	1
p	726838724295606890549323807888004534353641360687318060281490 199180612328166730772686396383698676545930088884461843637361 053498018365439

Using `SEA` algorithm:

```
sage: ec.order()
NP = 726838724295606890549323807888004534353641360687318060281490
019918058401584615834286478302116676950385324117483636664921909 5
023438599116
```

4 Curve Generation and Verification

Elliptic curves are the basis of an ECC. However, there is not a single standardized curve that is used throughout all ECC. There are atleast 8 standards that define different types of ECs. Each standard tries to ensure that ECC computations are efficient and ECDLP is hard [5].

There are total 20 curves that are evaluated by Dan Bernstein and Tanja Lange. These curves are determined as secure if they satisfy all the *SafeCurves* requirements. Each one of these curves have a different set of parameters for efficiency like different fields, shapes and size of cofactor. The curves that satisfy the `SafeCurves` criteria are gaining popularity in their usage. Curve25519 and Curve4448 are two such curves that are used in end-to-end encryption protocols, digital signature algorithms and key exchange protocols. Jubjub which is a twisted Edwards curve is used in crytpocurrencies like Zcash [12]. Due to their gaining popularity, studying the generation of these curves is important.

Wozny examines the domain parameters for generation of curves used in ECC, defined by National Institute of Standards and Technology (NIST), Institute of Electrical and Electronics Engineers (IEEE) standards, American National Standards Institute (ANSI), Secure and Efficient Cryptography Group (SECG) for binary Galois fields [25].

While Wozny discusses the curve generation and parameter validation of curves in Galois fields, a question that arises is, how to transparently generate curves that are secure? Transparent generation of curves mean that anyone can verify the curve parameters and there are no Nothing-up-my-sleeve numbers in those curve parameters.

Flori et al. [11] discuss this very question where they propose the a list of requirements that should be satisfied by an EC to be deemed as "secure". They also propose a method to generate and verify EC in a transparent method.

In this section we focus on the generation of "secure" elliptic curve. We study the cryptographic criteria that a curve should follow and propose our pair of "secure" curves. We implement the algorithm that can generate Montgomery curves which satisfy `SafeCurves` criteria. We explain the verification process of these curves using the `SafeCurves` verification tool.

4.1 Criteria for cryptographic ECs

To select a curve for cryptographic use, we have to consider the following criteria [11]

1. The ECDLP problem should be hard under the defined parameters of the curve.
2. EC should be such that it can be implemented to be resistant against side channel attacks.
3. Normality conditions should be satisfied by the curve in order to be secure against some particularly unknown attacks.
4. Implementation of the curve should be convenient.
5. Special families of curves used for specific protocols and algorithms.
6. Generation of the curve should be verifiable.

4.1.1 Hardness ECDLP problem

The security of the curve is determined by the hardness of ECDLP problem against known attacks. Flori et al. suggest some criteria required to generate a curve which which provides security against such known ECDLP attacks. The curve should be a non-singualar curve i.e the discriminant $4a^3 + 27b^2$ should be equal to 0. The order n of the curve should be a product of large prime number and a very small co-factor $h = n/p$. Small cofactor provides

security and performance improvements. Flori et al. recommend that the cofactor h should be 1, whereas there are other secure curves that have cofactor $h > 1$. It is recommended to define the curve over a prime field or binary Galois field $GF(2^m)$ in order to provide resistance against index calculus computations [11]. The cost of computing Pollard Rho should be greater that 2^{100}. ECDLP security does not ensure ECC security. There are other attacks that undermine the security of the curve while being ECDLP resistant [5].

4.1.2 Implementation dependent security

Improper implementation of the curve can make a cryptosystem vulnerable to side channel attacks. Even the choice of the curve can be a factor, as some curves are vulnerable to side channel attacks. Flori et al. suggest a few criteria to improve the security of the curve implementation. Namely, the curve should not have a small subgroup. Having *special points* on the curve can make the curve susceptible to side channel attacks. The *special points* of an EC are points (x, y) such that one of the two coordinates of is zero. The base field of the curve should not be a *special prime number*. There are some curves like Curve25519 or SM2 that are defined over a prime field $\mathbb{F}_p$, where p is a special prime number like pseudo-Mersenne or generalized Mersenne number. The fields based on these numbers have benefits like fast computations but they are susceptible to side-channel attacks [11]. `SafeCurves` require the base field to be prime and $p \equiv 1 \bmod 4$ or $p \equiv 3 \bmod 4$. Curves defined in *special prime number* field meet the `SafeCurves` criteria [5].

4.1.3 Normality of the curve

Flori et al. present the properties that random curves should satisfy with overwhelming probability. They insist these conditions does not make the curve secure against precise attacks, but if they are not met then it would make the curve slightly vulnerable. The conditions are:

- **Cardinality of the quadratic twist**
  The order of the quadratic twist has influence on the security of the curve. The order of quadratic twist should be large for the ECDLP problem to be hard.

- **Non-special base field**
  Special prime number is a number when it is a value of a polynomial of a low degree with small coefficients evaluated at a small value [11]. Flori et al. argue that there are no known attacks against curves with special parameters such as Curve25519 or FIPS 186-2 curves, it is legitimate to consider them as exceptional. However, using special primes can be advantageous in certain cases as they allow faster arithmetic operations. Hence, we will use special prime for generating Montgomery curves due the curve properties.

- **Embedding degree**
  Embedding degree of E is the smallest integer e such that q divides $p^e - 1$ where q is the largest prime divisor of the order of the curve n and p is the prime field. This is an expensive computation since we have to factor $q - 1$. While computing the twist security, we have to compute the embedding degree of the of the twist. Hence, we have to perform perform this task twice.

- **Multiplicative group of the base field**
  If $p - 1$ is smooth then the multiplicative discrete logarithm problem is easy. A number is smooth if its prime divisors are small. This computation is done only once.

- **Discriminant of the endomorphism ring**
  The discriminant of the endomorphism ring should be large. `SafeCurves` require the discriminant to be larger than 2^{100} [5].

- **Class number**
  Flori et al. suggest that the class number of the curve should be at least $p^{1/4}$ [11]. However, `SafeCurves` does not include class number requirement as it argues that class number can be derived from discriminant of the endomorphism ring. Hence, does not incorporate class number requirement [5].

The endomorphism of the curve gives us information of the structure of the curve. Curves with endomorphism rings with $\mathbb{Z}$-rank 4 are vulnerable curves [15]. The class number of the curve should also be large.

4.1.4 Convenience of implementation of the curve

The curve should be convenient to implement without affecting its security. If the number of points n is greater than p then it would be infeasible to represent $n-1$ numbers in the give memory space for a large p. If $p \equiv 3 \ mod \ 4$ then it would be efficient to use point compression method of representing points (x, y) of E. Selecting a special prime number can help in performing fast base field arithmetic but Flori et al. suggest that it would affect the optimal security of the curve. They recommenced using base fields which are more general. Using a special coefficient to perform fast arithmetic might be beneficial but it might be a security risk [11].

4.2 Generating an EC

Flori et al. propose a method to generate and validate a curve. They propose a program that generates a curve with parameters and another program that validates whether the parameters are cryptographically safe or not. They do not mention the method of checking the conditions that were mentioned in above sections. There are three conditions that are computationally expensive to check, which are checking the order of the curve and whether the curve has a small cofactor, computing the endomorphism ring and class number

and finally computing the embedding degree of the curve.

They present a toy example for generation and verification of the curve. Each curve would have a certificate that determines whether the curve is "good" or not. If the curve is good, then they present the proof of all the criteria of the curve.
Baier and Buchmann propose two methods to generate suitable EC group. Random approach, where the parameters of the EC are generated randomly and tested for their security and complex multiplication approach, which uses complex multiplication theory to generate a suitable group [2].
NIST SP 800-186 recommends a criteria that the curves should satisfy to be cryptographic. It proposes criteria for different families of curves like, weierstrass curves defined over prime and binary fields, Montgomery curves, Edwards curves and pseudo-random curves. We focus on security criteria of Montgomery and Edwards curve since they are used in `Signal` protocol. Edwards curve can be derived from Montgomery curve, hence our focus in this thesis will be generation of Montgomery curves.
We use the security criteria determined by NIST SP 800-186 to generate Montgomery curves. We analyse the performance for generating the curve parameters by using random approach. And finally we provide a "certification" of the security of the curve by using `Safecurves` verification tool.

4.3 Verification of an EC

Verification of ECs requires testing the parameters of curve for ECDLP and ECC security. Dan Bernstein and Tanja Lange have published a tool for verification of ECs. `Safecurves` criteria does not consider efficiency issues while verifying a curve. The authors argue that efficiency related requirements actually damage the efficiency and in some cases it is bad for the security of the curve [5]. We used this tool to verify the generated curve for its security. The tool is written in `Sage` and it takes input a directory that has multiple

files. Each file contains some information about the generated curves. The file names and information are as follows:

- p: the field that the curve is defined in (decimal).
- $|G|$: the prime order of the generator point Gs (decimal).
- $G(x1)$: x-coordinate of the G.
- $G(y1)$: y-coordinate of the G.
- $P(x)$: x-coordinate of a point P, where P generates the entire curve.
- $P(y)$: y-coordinate of a point P, where P generates the entire curve.
- shape: the curve shape, namely shortw or montgomery or edwards.
  - If shape is "shortw": a and b coefficients of weierstrass curve.
  - If shape is "montgomery": A and B coefficients of montgomery curve.
  - If shape is "edwards": d coefficient of the edwards curve.
- primes: all prime divisors of
  - prime field p
  - curve order $p + 1 - tr$, where tr is trace of elliptic curve
  - twist order $p + 1 + tr$
  - $tr^2 - 4 \cdot p$
  - recursive prime divisors of all $e - 1$ where e is the element in the list.

The program returns output with multiple files that helps us interpret the verification of elliptic curve. All files with name "verify-*" contain the

verification of properties of the curve. If the curve passes all the `SafeCurves` criteria, then the file "verify-safecurve" will have the value as **True**. Files with name "hex-*" contain hexadecimal representation of the parameters of the curve. Other files include proof of the primality of the numbers and representation of numbers in the power of 2.

4.4 Security criteria for Montgomery curves

Montgomery curve is another special type of which is used in ECC where fast x-point scalar multiplication is desired. It is defined by the equation: $E_{M,A,B} : By^2 = x^3 + Ax^2 + x \bmod p$, for $A, B \in p$ and $B(A^2 - 4) \neq 4$.

The value of the desired cofactor depends on the field p. If $p = 1 \bmod 4$, then the desired order of the curve and its twist are $\{4, 8\}$,$\{8, 4\}$. If we choose the first pair of cofactors, then the order of twist is greater than the order of the curve. Which would increase the computation for algorithms that take cofactors into account since might also check for points on the twist. Hence, we chose the cofactors $\{8, 4\}$.
If $p = 3 \bmod 4$, then the desired cofactor of twist and curve is $\{4, 4\}$ [18].
The requirements for the parameters A and B as per the NIST 186 standards are:

- The value of B should be 1.
- The value of A is selected as the minimum value where the following conditions should be satisfied:
  - The curve is cyclic implies that $A^2 - 4$ is not a square in GF(p).
  - The curve has cofactor of $h = 4$ or $h = 8$ implies that $A + 2$ is a square in GF(p).
  - The quadratic twist E' of the curve should have cofactor of $h = 4$.
  - A has the form $A \equiv 2 \bmod 4$.

- Select base point $G = (X_G, Y_G)$ such that $|X_G|$ is minimal and Y_G is odd.

We have used the above criteria and created an algorithm that can be used to generate secure Montgomery curves. The time complexity of the algorithm increases as the field k increases. The time complexity can be decreased further by using more processors. Note that this algorithm runs only once to get the desired parameters.

Algorithm 1: Generate A

```
Result: A
Initialize k = field
for A in range (1, k, 1) do
    if A%4 == 2 then
        if kronecker(A**2 − 4, k) == −1 then
            if kronecker(A + 2, k) == −1 then
                ec = EllipticCurve(GF(k), [0, A, 0, 1, 0])
                order = ec.order()
                factors = factor(ft)

                if (factors[0] == (2,2) and (len(factors) == 2) and
                 isprime(factors[1][0])) then

                    trace = k+1 - order
                    order_of_twist = k + 1 + trace
                    factor_of_twist = factor(order_of_twist)

                    if factor_of_twist[0] == (2,2) and
                     len(factor_of_twist) == 2 then
                        return A
                        break
                    end
                end
            end
        end
    end
end
```

One feature of Montgomery curve is its relationship with twisted Edwards curve. Ever Montgomery curve is birationally equivalent to twisted Edwards

curve and vice versa. We can derive a twisted Edwards curve from the any Montgomery curve using the given formula:

$$E_{E,a,d} \rightarrow \mathrm{E}_{M,A,B}: \quad A = \tfrac{2(a+d)}{a-d} \text{ and } B = \tfrac{4}{a-d}$$

$$(x, y) \rightarrow (u, v) = \left(\tfrac{1+y}{1-y}, \tfrac{1+y}{(1-y)x}\right).$$

Similarly,

$$E_{M,A,B} \rightarrow \mathrm{E}_{E,a,d}: \quad a = \tfrac{A+2}{B} \text{ and } d = \tfrac{A-2}{B}$$

$$(u, v) \rightarrow (x, y) = \left(\tfrac{u}{v}, \tfrac{u-1}{u+1}\right).$$

Using the above formula, we can derive any Edwards curve from a Montgomery curve.

4.5 Time analysis of computing A

The amount of time require to generate the parameter A is dependent on the size of the field p. As discussed in 4.4, the desired order of the curve when $p = 3 \bmod 4$ should be 4. Hence, we all the curves generated below have the cofactor 4 and twist cofactor also 4. The RFC 7748 emphasises on choosing the minimal value of A for performance and simplicity reasons. So we iterate from 3 to 10^9, to find the minimal value of A that satisfies the criteria discussed in 4.4. We observe that as the size of field increases, the time required to find the value of A also increases.

As discussed in 4.4, the desired order of the curve when $p = 1 \bmod 4$ should be 8. Hence, we all the curves generated below have the cofactor 8 and twist cofactor also 4. We observe that as the size of field increases, the time required to find the value of A also increases. The time required to compute A is smaller when $p = 1 \bmod 4$ compared to when $p = 3 \bmod 4$.

In 4.5, the value of A is calculated by iteratively searching from 3 to 10^9 such that it is minimal. However in some cases, finding the value of randomized A is takes less time compared to the iterative method. This does not ensure performance since the values of A are larger in size than iterative method.

Similar to the above, finding the value of randomized A in some cases take less time compared to the iterative method. As the size of field increases, the time required to find A is less than than the iterative method.

Field	Bits	Time (in sec)	A	h
599	10	0.0801	262	4
1283	11	0.0293	134	4
4079	12	0.024	130	4
46499	16	0.034	30	4
762871	20	0.057	274	4
1071919	25	0.387	2890	4
2835035807	32	0.204	730	4
298291166879	40	0.078	186	4
1043659579451143	45	1.44	2718	4
1043659579451143	50	2.38	1278	4
27523857120632423	55	35.89	9634	4
865827640841390683	60	17.63	8256	4
10480660404865665031	64	57.54	12694	4
426737804570514267864967	79	1038.27	65526	4

Table 1: Computing A iteratively, where $p = 3 \bmod 4$

Field	Bits	Time (in sec)	A	h
641	10	0.007	10	8
1601	11	0.162	1142	8
2749	12	0.190	226	8
7433	16	0.042	38	8
30389	20	0.181	538	8
12017497	25	0.655	3190	8
28043401	32	0.084	234	8
588858461273	40	0.496	838	8
13565825784053	45	1.145	1206	8
233132441592313	50	7.074	4738	8
10618124951016833	55	7.523	2390	8
903670300601356697	60	79.154	22750	8
18208804115091945829	64	32.510	6462	8
385274230067896555822949	79	76.053	3438	8

Table 2: Computing A iteratively, where $p = 1 \bmod 4$

Field	Bits	Time (in sec)	A	# of A's tried	h
46499	16	0.107	43798	461	4
762871	20	0.047	81578	186	4
1071919	25	0.068	349970	375	4
2835035807	32	0.357	2363480646	1060	4
298291166879	40	2.43	219835422630	3975	4
25157323951583	45	5.32	16092897198902	3762	4
1043659579451143	50	40.782	778655335914022	14137	4
27523857120632423	55	157.227	498878990099714	20835	4
865827640841390683	60	56.326	348429176562426818	17798	4
10480660404865665031	64	100.78	374416011502485650	21921	4
426737804570514267864967	79	58.541	180859438044348245316778	3000	4

Table 3: Computing A randomly, where $p = 3 \bmod 4$

Field	Bits	Time (in sec)	A	# of A's tried	h
641	10	0.131	90	53	8
1601	11	0.206	1390	1215	8
3461	12	0.024	3330	51	8
7433	16	0.022	5714	99	8
30389	20	0.041	810	321	8
12017497	25	0.122	7994794	622	8
28043401	32	0.337	15746794	1667	8
588858461273	40	1.485	229167062922	3407	8
13565825784053	45	4.0795	7939611849078	5425	8
233132441592313	50	1.360	64243845514354	1043	8
10618124951016833	55	32.318	174100300926558	9353	8
903670300601356697	60	55.836	188144657425353126	11173	8
385274230067896555822949	79	68.898	377311557684493588940514	2208	8

Table 4: Computing A randomly, where $p = 1 \bmod 4$

4.5.1 Iterative vs random search for A parameter

The proposed method in RFC 7748 for generating A, states that the value of A should be minimum such that it passes the necessary security criteria [18]. For a curve with field $p > 2^{100}$, the time required to find A iteratively is more than the finding A randomly.

For a field $p = 2^{89} - 1$, we get the value of A iteratively in 882.5 seconds while randomly finding A would take 167.7 seconds. Initially we find random element in the range of $(3, 10^9)$, then after getting the value we reduce the range again to find the next random value of A. We do this till we get the lowest value of A that matches with our iterative value. If we set the appropriate range then finding the value of A randomly is faster than the iterative method. This method gives us different values of A that can be used for generating ECC. However, the value of A, when generated randomly does not meet the aforementioned security criteria. But it can still be a used as an ephemeral curve.

Random generation method cannot be verified by any third party and the curve could be dismissed as suspicious. Using a CSPRNG can mitigate the suspicion for generating ephemeral curves.

Field	Bits	Time (in sec)	A	h
$2^{89} - 1$	89	882.535	32290	4
$2^{107} - 1$	107	*	*	4

Table 5: Iterative computation of A

Field	Bits	Time (in sec)	A	tries	Range
$2^{89}-1$	89	167.792	470642746	6269	$(3, 10^9)$
		1443.514	114920138	58867	$(3, 470642746)$
		87.067	34793102	3626	$(3, 114920138)$
		1336.258	59682	62398	$(3, 34793102)$
		497.891	32290	22523	$(3, 59682)$
$2^{107}-1$	107	151.403	806450534	3326	$(3, 10^9)$
		755.403	671948038	17705	$(3, 806450534)$
		4711.379	533718842	100001	$(3, 671948038)$
		255.761	41371402	6104	$(3, 533718842)$
		1704.339	16700578	46659	$(3, 41371402)$
		248.273	14663590	4235	$(3, 16700578)$
		2935.074	11784278	41908	$(3, 14663590)$
		1584.776	582346	42530	$(3, 11784278)$
		1465.827	211982	42339	$(3, 11784278)$

Table 6: Random computation of A

4.6 End-to-End Encryption Debate

`WhatsApp` is the most popular messaging application in the world with more than 1.5 billion users. Ever since `Facebook` had bought `WhatsApp` in 2014, concerns about privacy had increased amongst the tech community as `Facebook` does not have a stellar record in maintaining user privacy [10]. Forbes journalist Kalev Leetaru recently wrote about `Facebook's` plan to backdoor `WhatsApp` for client side content scanning and filtering. However, further inspection concluded that `Facebook` does not plan to implement a backdoor on `WhatsApp` at all and what Leetaru reported had very little to do

with WhatsApp and more about content filtering on Facebook using AI [20]. Facebook's malpractices with user privacy has put the company and its products under the scrutiny by privacy experts throughout the globe. The cryptographic protocols behind WhatsApp are deemed as secure and they do provide end-to-end encryption, however there are other issues that affect the user privacy. A backdoor that can record the user's screen activity can read the WhatsApp messages without even affecting the E2E protocol. Facebook or any highly motivated party can still read the WhatsApp messages due to improper implementation of user data storage [26]. While the underlying cryptosystems are secure, assuring privacy and security to the users under the guise of end-to-end encryption is a bit misleading. Better alternatives like Signal messenger should be used, which is an open source software unlike WhatsApp.

4.7 Contribution to thesis goals

In this section, we focus on goals G4 and G5. We have gone through all the necessary criteria required for a curve to satisfy to be deemed as secure. Adhering to these requirements, we would generate curve parameters and check if the curve passes all the necessary requirements. Once the generated curve passes all the necessary requirements, the next step would be to compute the order of the generated curve and finally test the ECDLP security of the curve.

5 Attacks on ECDLP

In this section we will focus on algorithms that solve ECDLP. Algorithms such as Shanks algorithm, Pollard rho algorithm and Pohlig-Hellman algorithm can solve the ECDLP problem in fields that can be considered small for cryptographic use.
While generating a secure EC, we have to choose the domain parameters such that ECDLP problem is hard. The algorithms that solve ECDLP should find it infeasible to solve ECDLP under the recommended parameters.

5.1 Shanks Algorithm

Shanks' Algorithm is a meet-in-middle algorithm to solve ECDLP. The algorithm works as follows [13]:

1. Pick an integer $m = \lceil \sqrt{n} \rceil$
2. Compute mP
3. for $i \leftarrow 0$ to $m - 1$ compute and store iP
4. for $j \leftarrow 0$ to $m - 1$ compute and store $Q - jmP$
5. Sort both the lists
6. Search through both the lists to find collision such that $iP = Q - jmP$

The collision is our desired solution. The space and time complexity of this algorithm is $\mathcal{O}(\sqrt{n})$.
A modification to Shanks' baby step giant step (BSGS) algorithm is proposed by Bernstein and Lange [6].

5.2 Pollard Rho

As discussed in section 5.1, Shanks algorithm has space and time complexity of $\mathcal{O}(\sqrt{n})$. For large numbers, this algorithm becomes expensive to compute and store. Pollard rho is an improvement over Shanks as it requires less storage compared to Shanks and can be parallelised [13]. Pollard rho can be used to solve DLP and ECDLP problems. In this section we will focus on Pollard rho for attacking ECDLP.
The algorithm works as follows [7]:

1. Partition the curve E (points on the curve) defined over $\mathbb{F}_{2^k}$ into three equal subsets of S such that $S_1 \cup S_2 \cup S_3 = S$

2. Blumenfeld suggests that we can do step 1 by either reducing x-coordinate modulo 3 or use projective coordinate and reduce y-coordinates modulo 3 [7].

3. Now select a random base point α modulo n such that $A_0 = \alpha P$ (Refer 5 for P)

4. $A_{i+1} = \mho(A_i) = \begin{cases} A_i + P \text{ if } A_i \in S_1, \\ 2A_i \text{ if } A_i \in S_2, \\ A_i + Q \text{ if } A_i \in S_3. \end{cases}$

   The sequence A_i takes up the form $A_i = a_i P + b_j Q$

5. If $A_{i1} = A_{i2}$, then we get $a_{j1}P + b_{j1}Q = a_{i2}P + b_{j2}Q$

6. We get $\frac{a_{j1}-a_{j2}}{b_{j1}-b_{j2}}P = Q$

7. if $gcd(b_{j2} - b_{j1}, n) = 1$, then $\frac{a_{j1}-a_{j2}}{b_{j1}-b_{j2}}$ is easy to calculate.
   if $gcd(b_{j2} - b_{j1}, n) = d > 1$, then we can compute $\frac{a_{j1}-a_{j2}}{b_{j1}-b_{j2}} \mod (N/d)$.

The above algorithm for curves defined over field $\mathbb{F}_{2^k}$ but it can be extended to the curves defined over a prime field $\mathbb{F}_p$.
The Pollard rho can be enhanced or its speed can be increased using different concepts. Brent's cycle finding algorithm speeds up the algorithm by 24%. Pollard rho can be parallelized using multiple processors. Automorphism can be also be used to speed up the algorithm. Faster and efficient algorithms can be used to save a lot of gcd computations [8].

The sequences made in the above algorithm are called walks. There has been subsequent amount of research done to change the number of walks and examine the complexity of the algorithm. Kangaroo method and Gaudry-Schost are different versions of Pollard rho algorithm. Kangaroo method is suitable for intervals of N (where N $<$ ord(P)). Compared to pollard rho, the steps in pseudorandom walks are small. Gaudry-Schost method is a combination of Pollard rho and Kangaroo method where the algorithm uses small jumps and is analysed using the birthday paradox [13].
`SafeCurves` criteria require the order of the curve to be greater than 2^{200}. The most efficient Pollard rho algorithm would require around $0.886 * \sqrt{order}$ times addition operations [5]. By using this criteria, we can be certain that no curve would be vulnerable to pollard rho attack for a distant future.

5.3 Pohlig-Hellman

Pohlig-Hellman algorithm solves the DLP problem by reducing the problem into prime subgroups of P [22]. The best known attack on ECDLP is by using a combination of Pohlig-Hellman and Pollard rho algorithm. The worse case complexity of Pohlig-Hellman algorithm is $\mathcal{O}(\sqrt{n})$. The order of the base point $|G|$ is divisible by the order of the curve. For the curve to be secure, the order of the base point $|G|$ should be greater than 2^{200}. Pohlig-Hellman finishes faster if the the order $|G|$ has prime factors. The base points of Curve272 and Curve224 are prime and greater than 2^{200}, hence they are

resistant against Pohlig-Hellman attack.

5.4 Experimental work

We implement Pollard rho algorithm in `SageMath` to solve ECLDP. To test the correctness of our implementation, we take a small Montgomery curve $y^2 = x^3 + 54x^2 + x \mod 8191$. Consider a point P and Q on the curve such that $P = mQ$, where m is a random number. Using our implementation of Pollard rho, we have to find m. To run this experiment, we take a random point P and Q and find the arbitrary m. Once we find m, we verify $P = mQ$. The following code returns the value of $m = 354$, which is correct.

```
ec = EllipticCurve(GF(8191), [0,54,0,1,0])
n = ec.order()
G = ec([4,3361,1])
P = ec([3678,2464,1])
Q = ec([3969,3046,1])

S1=[]
S2=[]
S3=[]

def f(x,a,b,P,Q,n,G):
    if f_bucket(x) ==  "S1":
        return [x+P ,a % n,(b+1) % n]
    elif f_bucket(x) ==  "S2":
        return [x+x,(2*a)%n, (2*b)%n]
    elif f_bucket(x) ==  "S3":
        return [Q+x, (a+1)%n, b%n]

def pollard_rho(G,n,P,Q):
```

```
[x,a,b] = f(K, 0, 0, P, Q, n,G)
[x2,a2,b2] = f(x,a,b,P,Q,n,G)

while((a*P + b*Q)!=(a2*P + b2*Q)):

    [x,a,b] = f(x, a, b, P, Q, n,G)
    [x2,a2,b2] = f(x2, a2, b2, P, Q, n,G)
    [x2,a2,b2] = f(x2, a2, b2, P, Q, n,G)

if gcd(b2-b,n) != 1:
    return "failure"
else :
    return  (mod((a-a2),n)* inverse_mod((b2-b),n))%n
```

5.5 Contribution to thesis goals

This section focuses on goal G5. The safe curves we generated are resistant against ECDLP solving algorithms like Pollard rho and Pohlig-Hellman.

Part II

Case study of elliptic curves

6 Generated curves

6.1 Curve224

Motivation:

In the above examples, we have generated curves that are insecure and not suitable for cryptography purpose. In this section, we propose a Curve224 which can be used in ECC. Due to its security strength, it cannot be used in `Signal` protocol, but it can be used for other ECC applications that use ECDH and ECDSA. There is a Weierstrass curve nistp224 approved by NIST that is defined over the same field. Nistp224 is not considered safe as per the `SafeCurves` criteria. The generation of nistp224 parameters are suspicious [5]. Hence, we want to propose a Montgomery curve in the same field but which passes the `SafeCurves` criteria.

Field choice:

We use a NIST approved Solinas prime number $2^{224} - 2^{96} + 1$. We use the same field to generate a more secure EC. This prime is 224 bits long, hence it can be efficiently represented as a multiple of computer word size [16]. This is helpful in generation of crytographic protocols.

Curve coefficients:

We generate a Montgomery curve of the form:

$$By^2 = x^3 + Ax^2 + x$$

The parameter B is set to 1. The value of A is 64486, which is the minimum value that satisfies all the criteria. We can verify that A is a nothing-up-my-sleeve number using the algorithm 1. This curve is satisfies all the `SafeCurves` criteria.
The order of the curve is

$$8 \cdot n = 8 \cdot \left(2^{221} - 1207077220185791217085940763318325\right)$$

and the order of twist is

$$4 \cdot n = 4 \cdot \left(2^{222} + 24141148262903253020030847546614 83\right).$$

The generator G of the curve should be a point with lowest possible u-coordinate. Using the code from RFC 7748, we get the generator of the curve as

$$\left(3, 20130223139539004842640525492283454016648958047394819699224 1577 330\right).$$

SafeCurves verification

We use `SafeCurves` verification tool to demonstrate the security of the Curve224. The curve passes all the security criteria. We can demonstrate the transparent generation of the curve using the algorithm 1. The output of the tool can be found here.

Field	p prime	True
Equation	$y^2 = x^3 + 64486x^2 + x$	True
Base	G on curve	True

Table 7: Parameters

Rho	rho above 2^{100}	True
Transfer	safe against additive/multiplicative transfer	True
Discriminants	$\|D\|$ above 2^{100}	True
Rigid	rigid	True

Table 8: ECDLP security

Ladder	montgomery ladder	True
Twist	cost of combined attack above 2^{100}	True
Completeness	complete single/multi-scalar formulas	True
Indistinguishability	supports indistinguishability	True

Table 9: ECC security

6.2 Curve272

Motivation:

Curve224 satisfies the `SafeCurves` criteria, however it is less secure compared to Curve25519. Our main goal of this thesis is to generate a curve that is secure and can be used as an alternative EC for cryptography purpose especially in `Signal` protocol. The two existing curves that are already used

in `Signal` are Curve25519 and Curve448. The former is used widely in other ECC while the latter is considered as an "overkill" curve due to its large field. We propose a new curve in order to diversify the choices of elliptic curves. In order to generate this curve, we follow the *SafeCurves* security criteria and the NIST 186 standards.

Field choice:

In order to generate a secure curve, we need to define the curve in a large field such that the ECDLP problem is hard to solve. We choose the prime field $p = 2^{272} - 2^{40} - 1$, which is a `Solinas` prime number. `Solinas` prime numbers are widely used in cryptography, especially in ECC [16].

Curve coefficients:

We generate a Montgomery curve of the form:

$$By^2 = x^3 + Ax^2 + x$$

The parameter B is set to 1. The value of A is 22042, which is the minimum value that satisfies all the criteria. We can verify that A is a nothing-up-my-sleeve number using the algorithm 1. This curve is satisfies all the `SafeCurves` criteria.
The order of the curve is

$$4 \cdot n = 4 \cdot \left(2^{270} + 924405772094358431727659602441667667558 9\right)$$

and the order of twist is

$$4 \cdot n = 4 \cdot \left(2^{270} - 924405772094358431727659602496643248947 7\right).$$

The generator G of the curve should be a point with lowest possible u-coordinate. Using the code from RFC 7748, we get the generator of the curve as

$$\Big(2, 6225892792642318762314422867744054013250329433824992078314983515034034530871067 01\Big).$$

SafeCurves verification:

We use `SafeCurves` verification tool to demonstrate the security of the Curve272. Our curve passes all the security requirements and is determined as "safe". The output of the tool can be found here.

Field	p prime	True
Equation	$y^2 = x^3 + 22042x^2 + x$	True
Base	G on curve	True

Table 10: Parameters

Rho	rho above 2^{100}	True
Transfer	safe against additive/multiplicative transfer	True
Discriminants	$\|D\|$ above 2^{100}	True
Rigid	rigid	True

Table 11: ECDLP security

Ladder	montgomery ladder	True
Twist	cost of combined attack above 2^{100}	True
Completeness	complete single/multi-scalar formulas	True
Indistinguishability	supports indistinguishability	True

Table 12: ECC security

6.3 Toy EC

To demonstrate the criteria for generation of curves, we create a small EC with crytographically insignificant parameters. The process to generate crytographically secure curves is the same, except the time required to generate and verify the parameters will vary.

6.3.1 Curve31

Generating a Montgomery curve based on the NIST criteria is essential for its security. We choose a field $\mathbb{F}_p$ with $p = 2^{31} - 1$, which is 32 bits long Mersenne prime number. The reason we chose this field is because it is not computationally expensive to run experiments such as finding A parameter or computing all private keys. The value of B is 1 and we have to find the minimum value of A that satisfies all the security criteria. We have the curve

$$E_{M,A,B} : y^2 = x^3 + 6222x^2 + x \bmod 2147483647.$$

The curve has the cofactor $h = 4$ and the order of the curve is 536855567 which is prime. The cofactor of the quadratic twist of $E_{M,A,B}$ is 4. Using the birational maps, we generate Ed32

$$E_{E,a,d} : 6224y^2 + y^2 = 1 + 6220x^2y^2 \bmod 2147483647.$$

We will use Curve32 to demonstrate ECDL attacks on elliptic curves.

6.3.2 Curve61

We choose a field $\mathbb{F}2^{61} - 1$, which is 61 bits long Mersenne prime number. The value of B is 1 and we have to find the minimum value of A that satisfies all the security criteria. We have the curve,

$$E_{M,A,B} : y^2 = x^3 + 50042x^2 + x \bmod 2305843009213693951$$

The curve has the cofactor $h = 4$ and the order of the curve is 576460752120095597 which is prime. The cofactor of the quadratic twist of $E_{M,A,B}$ is 4. Using the birational maps, we generate Ed61

$$E_{E,a,d} : 50044y^2 + y^2 = 1 + 50040x^2y^2 \bmod 2305843009213693951.$$

6.3.3 Ed1051

Twisted Edwards curve is a special type of elliptic curve that is used in modern ECC. It has desired properties such as fast explicit addition and doubling operations and birational relationship with other special curves.
Twisted Edwards curve is defined by the equation $E_{E,a,d} : ax^2+y^2 = 1+dx^2y^2$ over a non binary field k. The point addition formula is complete if a is a square in k and d is a non-square in k. Using this criteria, we generate a toy curve for field $p = \mathbb{F}1051$. Note that an Edwards curve is just an twisted Edwards curve with $a = 1$.
Using the defined conditions we generate a toy curve: $-3x^2 + y^2 = 1 + 7x^2y^2$ mod 1051. We can count the points on the curve using trivial brute force method, but for large fields we will use Schoof's or SEA algorithm.

6.4 Toy ECC

We use the generated Curves to construct toy cryptosystem. We choose to construct a toy Elliptic curve Diffie-Hellman(ECDH) cryptosystem because it is used in `Signal` protocol. Diffie-Hellman key exchange requires fast curve operations, hence implement birational mapping to convert our Montgomery curve to Edwards curve. The code for this ECC can be found in appendix

6.4.1 EC arithmetic:

We implemented Edwards and Montgomery curve point arithmetic in C++ using GMP library. The point arithmetic of twisted edwards curve is *complete*.

Hence, it is a preferred choice for digital signature algorithms.

Edwards curve point addition

The point arithmetic formulas can be found in section 2.5.2

```
//Returns R = P+Q

void add(R,P,Q,ECurve E){


    Initialize xtop,ytop,xtemp,ytemp;


    xtop = (P.X * Q.Y) + (Q.X * P.Y);
    xtemp = 1 + d * P.X * P.Y * Q.X * Q.Y;
    invert(xtemp,field);
    xtop = xtop*xtemp;

    ytop = (P.Y * Q.Y) - (a * P.X * Q.X);
    ytemp = 1 - d * P.X * P.Y * Q.X * Q.Y;
    ytop = ytop*ytemp;


    swap(R.X,xtop);
    swap(R.Y,ytop);

}
```

To demonstrate the output of the following pseudocode. Consider two points P and Q, we will add them to get the point R, $R = P + Q$. For the algorithm to be correct, the point R should lie on the curve.

$P = (196822117242893674459907784170138183581511786955691996189713916913$

94964886553, 463168356949264781694283940034751631413079938662562256157830
33603165251855960)

Q = (429282026443452444960986984045526383989547290543233192385912667476
02475846613, 342488378102318364223308361101065469819997782266143592323114
90061590115997010)

$R = P+Q$ = (422744914654425032788592804291108266031846638811930885879496
34779277119126512, 464596337676635957844047528609053781596479806341115261
6228797618401576575679 9)

Edwards curve point doubling

The point arithmetic formulas can be found in section 2.5.2

```
//Returns R = 2*P

void ecdouble(R,P,ECurve E){
    add(R,P,P,E);
}
```

To demonstrate the output of the following pseudocode, we take the above point P and double it. The point $2P$ should lie on the curve to demonstrate the correctness of the algorithm.

P = (196822117242893674459907784170138183581511786955691996189713916913
94964886553, 463168356949264781694283940034751631413079938662562256157830
33603165251855960)

$2 * P$ = (327927243224797030249662180911230684369475216095591844677755412 66
189550147464, 155496755802801901763526687104495422515495720664450605805 07
079593062643049417)

Scalar multiplication

We implement double-and-add algorithm for scalar multiplication. This algorithm is the simplest scalar multiplication algorithm amongst all other methods. This algorithm calculates $n * P$ using systematic point doubling and addition techniques.

```
// Returns R = x*P

//core sclar multiplication
scalarmult(R,P,x,ECurve E){

  if(x == 0){
      R = identity_element;
      return R;
    }

  R = scalarmult(R,P,x/2,E);

  add(R,R,R,E);

  if (x % 2 == 1){
    add(R,R,P,E);
    }
  return R;
}

//Helper function
void scalar_mult(R,P,x,ECurve E){
     R = scalarmult(R,P,n,E);

}
```

To demonstrate the correctness of the pseudocode, we take the above point P and

multiply it with a random scalar value $x = 5412$. The point xP should lie on the curve to demonstrate the correctness of the algorithm.

$P = (196822117242893674459907784170138183581511786955691996189713916913$
$94964886553, 463168356949264781694283940034751631413079938662562256157830$
$33603165251855960)$

$5412 * P = (118536300272014934220347341623102204120108607734740086416964520$
$02476383726875, 153216758645389586125817851294682459095035829888157079589$
$907554525277450765236)$

6.4.2 Montgomery ladder

Montgomery ladder is a secure and side channel resistant scalar multiplication algorithm. This algorithm computes a fixed number of steps regardless of the value of scalar n. This property does not reveal any side channel information like power or timing. Due to this property, it is widely used in key generation and key exchange algorithms. As the name suggests, montgomery ladder can be efficiently implemented using montgomery curves. The montgomery point addition and doubling code can be found in appendix.

```
//Returns R = x*P
void Mladder(R,P,x,MontCurve M){

  Point R0("0","1");
  Point R1 = P;
  Point temp0,temp1;

  //value of x in binary
  value_str = str(x,2)
```

```
  for (i = 0 ; i <= value_str.length()-1  ; i++){
      if(arr[i] == 0){
          M.add(temp1,R1,R0,M);
          M.ecdouble(temp0,R0,M);

          R1 = temp1;
          R0 = temp0;
        }

      else{
          M.add(temp0,R0,R1,M);
          M.ecdouble(temp1,R1,M);

          R0 = temp0;
          R1 = temp1;
        }
    }
   R = R0;
}
```

To demonstrate the montgomery ladder, we take the generator G of Curve25519 and multiply it by some arbitrary scalar 4541. The output point should lie on the curve.

$G = (9, 43114425171068552920764898935933967039370386198203806730763910166200978582548)$

$4541 * G = (56284316850606511361097558627411582083904015489342316728361590610789917865926, 18247855913749052009798435451104351729088850768055439592350004675399403713941)$

6.4.3 Point compression

Point compression is a method to represent the EC points in a compact way. We have seen the representation of points in (x, y) coordinate system, but we can also represent elliptic curve points by removing the y coordinate. This is due to the property of elliptic curves defined over finite field. For every x coordinate, there are two y coordinates. One of them is even and the other is odd. So we can represent any EC point in a compressed format as $(x, 0/1)$ where 0 is for even coordinate and 1 is for odd coordinate [17].

```
\\Input (x,y)
\\Output (x,0/1)

String (Point G){
    if (y%2 == 0){
       return str(x,0)
       }
    else if(y%2 == 1){
       return str(x,1)
       }

}
```

Take the generator G of Curve25519. The y coordinate of G is even so it can be represented as $(x, 0)$:

$G = (9, 43114425171068552920764898935933967039370386198203806730763910166200978582548)$

$G = (9, 0)$
The main benefit of point compression is that the keys of EC can be represented in a smaller size without compromising the security. Point decompression is the

method of deriving the original point from a compressed point. Based on the y coordinate, we can derive the original point of the curve.

```
\\Input (x,0/1)
\\Output (x,y)

String (Point compressed_G){

t = x^3 + Ax^2 + x

if t is a square in F{
       if(x,0){
          y = mod(t,p).sqrt()
        }
       else{
          y = mod(t-p,p).sqrt()
          }
    }
    return (x,y)
}
```

$G = (9, 0)$

$G = (9, 43114425171068552920764898935933967039370386198203806730763910166200978582548)$

7 Standardized Curves

Several organizations define elliptic curve domain parameters and properties in order to determine their security. Each standard proposes a set of rules to generate curves and test vectors to verify them. Over the course of years, the organizations have refined their standards to be well accepted by the cryptography community. Bernstein and Lange argue that secure implementation of standard curve is theoretically hard and none of the standards ensure a good job of providing ECC security. There are other attacks that break ECC without solving ECDLP [5].

7.0.1 Curve catalog

Weierstrass Curves

Curve	Bit strength	Comments
P-192	96 bits	Legacy use only
P-224	112 bits	
P-256	128 bits	
P-384	192 bits	
P-521	261 bits	
W-25519	128 bits	Weierstrass form of Curve25519
W-448	224 bits	Weierstrass form of Curve448

Twisted Edwards Curves

Curve	Bit strength	Comments
Edwards25519	128 bits	Used in Signal Protocol and EdDSA
Edwards448	224 bits	Used in Signal Protocol and EdDSA
E448	224 bits	Used in Signal Protocol and EdDSA. Edwards448 is 4-isogenous to curve E448

Montgomery Curves

Curve	Bit strength	Comments
Curve25519	128 bits	Used in Signal Protocol
Curve448	224 bits	Used in Signal Protocol

Koblitz curves

Curve	Bit strength	Comments
Curve K-163	82 bits	For legacy use only
Curve K-283	142 bits	
Curve K-409	205 bits	
Curve K-571	286 bits	

Pseudorandom curves

Curve	Bit strength	Comments
Curve B-163	82 bits	For legacy use only
Curve B-283	142 bits	
Curve B-409	205 bits	
Curve B-571	286 bits	

7.0.2 NIST SP 800-186

NIST SP 800-186 is a standard proposed by NIST for Digital Signtaures and Ellipitic curve cryptography. This standard adds two new curves, Ed448 and Ed25519 for EdDSA use. In addition, the standard advocates the use of ECDSA and removal of DSA citing security analysis and growing popularity of ECDSA. The NIST curves proposed in the document are as follows:

Weierstrass Curves

Weierstrass curves that are generated over prime fields P-192, P-224, P-256, P-384,

and P-521 which have equation $y^2 = x^3 + ax + b$.

Curve	a	b	p
P-224	-3	18958286285566608000 40866854449392641550 46809686793210757872 34672564	$2^{224} - 2^{96} + 1$
P-256	-3	41058363725152142129 32612978004726840911 44410159937255548352 56314039467401291	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
P-384	-3	27580193559959705877 84901184038904809305 69058563615685214227 58019355995970587784 90118403890480930569 0585636156852142	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
P-521	-3	10938490380737342745 11112390766805569936 20759895168374899458 63944959531161507350 16013708737573759623 24859213229670631330 94384525315910129121 42327488478985984	$2^{521} - 1$
W-25519	19298681539552699237261830834781317975544974442734273399095973345732416392 36	55751746669818908907 64528907825714081824 11037279010123152944 00837956729358436	$2^{255} - 19$

W-448	484559149530404 593699549205258 669689569094240 458212040187660 132787074885444 487181790930922 465784363953393 589641229091574 035657199637535	26919952751689144094 419400292148316087171 90224767844667709222 95992819380802492878 77273940136988020219 63292164673494953191 91685664513904	$2^{448} - 2^{224} - 1$

Twisted Edwards Curves

Twisted Edwards Curve have equation $ax^2 + y^2 = 1 + dx^2y2$

Curve	a	d	p
Edwards25519	-1	$-121665/121666$	$2^{255} - 19$
Edwards448	1	-39081	$2^{448} - 2^{224} - 1$
E448	1	$39082/39081$	$2^{448} - 2^{224} - 1$

Montgomery Curves

Montgomery curves have equation $By^2 = x^3 + Ax^2 + x$

Curve	A	B	p
Curve25519	486662	1	$2^{255} - 19$
Curve448	156326	1	$2^{448} - 2^{224} - 1$

Koblitz Curves

Koblitz curve has equation $y^2 + xy = x^3 + ax^2 + b$

Curve	a	b	f(z)
Curve K-163	0	1	$z^{233} + z^{74} + 1$
Curve K-283	0	1	$z^{283} + z^{12} + z^7 + z^5 + 1$
Curve K-409	0	1	$z^{409} + z^8 + 1$
Curve K-571	0	1	$z^{571} + z^{10} + z^5 + z^2 + 1$

Pseudorandom Curves

Pseudorandom curve has equation $y^2 + xy = x^3 + x^2 + b$

Curve	a	b	f(z)
Curve B-163	1	0x066647EDE6C332C7F8C0923BB 58213B333B20E9CE4281FE115F7 D8F90ADA581485AF6263E313B79 A2F5	$z^{233} + z^{74} + 1$
Curve B-283	1	0x27B680AC8B8596DA5A4AF8A1 9A0303FCA97FD7645309FA2	$z^{283} + z^{12} + z^7 + z^5 + 1$
Curve B-409	1	0x021A5C2C8EE9FEB5C4B9A753 B7B476B7FD6422EF1F3DD674761 FA99D6AC27C8A9A197B272822F 6CD57A55AA4F50AE317B13545F	$z^{409} + z^8 + 1$
Curve B-571	1	0x2F40E7E2221F295DE297117B7F 3D62F5C6A97FFCB8CEFF1CD6B A8CE4A9A18AD84FFABBD8EFA 59332BE7AD6756A66E294AFD185 A78FF12AA520E4DE739BACA0C 7FFEFF7F2955727A	$z^{571} + z^{10} + z^5 + z^2 + 1$

7.0.3 SEC 2

SEC 2 is a standard proposed by Certicom Research for elliptic curves domain parameters. The SEC-2 curves proposed in the document are as follows:

192-bit Domain Parameters over $\mathbb{F}_p$
Parameters of secp192k1 are associated with a Koblitz curve, and secp192r1 are associated with verifiably random parameters.

secp192k1

p	$2^{192} - 2^{32} - 2^{12} - 2^8 - 2^7 - 2^6 - 2^3 - 1$
a	0x000000000000000000000000000000000000000000000000
b	0x000000000000000000000000000000000000000000000003
h	1
n	0xFFFFFFFFFFFFFFFFFFFFFFFE26F2FC170F69466A74DEFD8D

secp192r1

p	$2^{192} - 2^{64} - 1$
a	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC
b	0x64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
h	1
n	0xFFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831

224-bit Domain Parameters over $\mathbb{F}_p$

Parameters of secp224k1 are associated with a Koblitz curve, and secp224r1 are associated with verifiably random parameters.

secp224k1

p	$2^{224} - 2^{32} - 2^{12} - 2^{11} - 2^9 - 2^7 - 2^4 - 2 - 1$
a	0x00000000000000000000000000000000000000000000000000000000
b	0x00000000000000000000000000000000000000000000000000000005
h	1
n	0x0100000000000000000000000001DCE8D2EC6184CAF0A971769FB1F7

secp224r1

p	$2^{224} - 2^{96} + 1$
a	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFE
b	0xB4050A850C04B3ABF54132565044B0B7D7BFD8BA270B39432355FFB4
h	1
n	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF16A2E0B8F03E13DD29455C5C2A3D

256-bit Domain Parameters over $\mathbb{F}_p$

Parameters of secp256k1 are associated with a Koblitz curve, and secp256r1 are associated with verifiably random parameters. Secp256k1 is a curve used in Bitcoin cryptocurrency system. It is chosen because of its efficient and fast computation properties. The parameters of Secp256k1 are verifiable and selected in a predicted way such that it eliminates a possibility of having any backdoor.

secp256k1 (Bitcoin curve)

p	$2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
a	0x0000000000000000000000000000000000000000000000000000000000000000
b	0x0000000000000000000000000000000000000000000000000000000000000007
h	1
n	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BB FD25E8CD0364141

secp256r1

p	$2^{224}(2^{32} - 1) + 2^{192} + 2^{96} - 1$
a	0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFF FFFFFFFC
b	0x5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3 C3E27D2604B
h	1
n	0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9 CAC2FC632551

384-bit Domain Parameters over $\mathbb{F}_p$

secp384r1 curve is associated with verifiably random parameters.

secp384r1

p	$2^{384} - 2^{128} - 2^{96} + 232 - 1$
a	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFEFFFFFFFF0000000000000000FFFFFFFC
b	0xB3312FA7E23EE7E4988E056BE3F82D19181D9C6EFE8141120314088F5 013875AC656398D8A2ED19D2A85C8EDD3EC2AEF
h	1
n	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC 7634D81F4372DDF581A0DB248B0A77AECEC196ACCC52973

521-bit Domain Parameters over $\mathbb{F}_p$

secp521r1 curve is associated with verifiably random parameters.

secp521r1

p	$2^{521} - 1$
a	0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC
b	0x0051953EB9618E1C9A1F929A21A0B68540EEA2DA725B99B315F3B8B4 89918EF109E156193951EC7E937B1652C0BD3BB1BF073573DF883D2C34 F1EF451FD46B503F00
h	1
n	0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFA51868783BF2F966B7FCC0148F709A5D03BB5 C9B8899C47AEBB6FB71E91386409

163-bit Elliptic Curve Domain Parameters over $\mathbb{F}_{2m}$

Parameters of sect163k1 are associated with a Koblitz curve, sect163r1 and sect163r2 are associated with verifiably random parameters.

sect163k1

$f(x)$	$x^{163} + x^7 + x^6 + x^3 + 1$
a	0x000000000000000000000000000000000000000000000001
b	0x000000000000000000000000000000000000000000000001
h	2
n	0x04000000000000000000020108A2E0CC0D99F8A5EF

sect163r1

$f(x)$	$x^{163} + x^7 + x^6 + x^3 + 1$
a	0x07B6882CAAEFA84F9554FF8428BD88E246D2782AE2
b	0x0713612DCDDCB40AAB946BDA29CA91F73AF958AFD9
h	2
n	0x03FFFFFFFFFFFFFFFFFFF48AAB689C29CA710279B

sect163r2

$f(x)$	$x^{163} + x^7 + x^6 + x^3 + 1$
a	0x000000000000000000000000000000000000000000000001
b	0x020A601907B8C953CA1481EB10512F78744A3205FD
h	2
n	0x040000000000000000000292FE77E70C12A4234C33

233-bit Elliptic Curve Domain Parameters over $\mathbb{F}_{2m}$

Parameters of sect233k1 are associated with a Koblitz curve, sect233r1 are associated with verifiably random parameters.

sect233k1

$f(x)$	$x^{233} + x^{74} + 1$
a	0x000000000000000000000000000000000000000000000000000000000000
b	0x000000000000000000000000000000000000000000000000000000000001
h	4
n	0x8000000000000000000000000000069D5BB915BCD46EFB1AD5F173ABDF

sect233r1

$f(x)$	$x^{233} + x^{74} + 1$
a	0x000000000000000000000000000000000000000000000000000000000001
b	0x0066647EDE6C332C7F8C0923BB58213B333B20E9CE4281FE115F7D8F90AD
h	2
n	0x0100000000000000000000000000013E974E72F8A6922031D2603CFE0D7

239-bit Elliptic Curve Domain Parameters over $\mathbb{F}_{2m}$

Parameters of sect239k1 are associated with a Koblitz curve, sect239r1 are associated with verifiably random parameters.

sect239k1

$f(x)$	$x^{239} + x^{158} + 1$
a	0x000000000000000000000000000000000000000000000000000000000000
b	0x000000000000000000000000000000000000000000000000000000000001
h	4
n	0x2000000000000000000000000005A79FEC67CB6E91F1C1DA800E478A5

283-bit Elliptic Curve Domain Parameters over $\mathbb{F}_{2m}$

Parameters of sect283k1 are associated with a Koblitz curve, sect283r1 are associated with verifiably random parameters.

sect283k1

$f(x)$	$x^{283} + x^{12} + x^7 + x^5 + 1$
a	0x000000000000000000000000000000000000000000000000000000000000000000000000
b	0x000000000000000000000000000000000000000000000000000000000000000000000001
h	4
n	0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE9AE2ED07577265DFF7F94451E061E163C61

sect283r1

$f(x)$	$x^{283} + x^{12} + x^7 + x^5 + 1$
a	0x00000000000000000000000000000000000000000000000000000000000000000000000000000001
b	0x027B680AC8B8596DA5A4AF8A19A0303FCA97FD7645309FA2A581485AF6263E313B79A2F5
h	2
n	0x03FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEF90399660FC938A90165B042A7CEFADB307

409-bit Elliptic Curve Domain Parameters over $\mathbb{F}_{2m}$

Parameters of sect409k1 are associated with a Koblitz curve, sect409r1 are associated with verifiably random parameters.

sect409k1

$f(x)$	$x^{409} + x^{87} + 1$
a	0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
b	0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
h	4
n	0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE5F83B2D4EA20400EC4557D5ED3E3E7CA5B4B5C83B8E01E5FCF

sect409r1

$f(x)$	$x^{409} + x^{87} + 1$
a	0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
b	0x0021A5C2C8EE9FEB5C4B9A753B7B476B7FD6422EF1F3DD674761FA99D6AC27C8A9A197B272822F6CD57A55AA4F50AE317B13545F
h	2
n	0x010000000000000000000000000000000000000000000001E2AAD6A612F33307BE5FA47C3C9E052F838164CD37D9A21173

571-bit Elliptic Curve Domain Parameters over $\mathbb{F}_{2m}$

Parameters of sect571k1 are associated with a Koblitz curve, sect571r1 are associated with verifiably random parameters.

sect571k1

$f(x)$	$x^{571} + x^{10} + x^5 + x^2 + 1$
a	0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
b	0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
h	4
n	0x0200000000000000000000000000000000000000000000000000000000000000000000000000000000000131850E1F19A63E4B391A8DB917F4138B630D84BE5D639381E91DEB45CFE778F637C1001

sect571r1

$f(x)$	$x^{571} + x^{10} + x^5 + x^2 + 1$
a	0x0000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000000 00000000000000000001
b	0x02F40E7E2221F295DE297117B7F3D62F5C6A97FFCB8CEFF1CD6B A8CE4A9A18AD84FFABBD8EFA59332BE7AD6756A66E294AFD185A 78FF12AA520E4DE739BACA0C7FFEFF7F2955727A
h	2
n	0x03FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFFFFE661CE18FF55987308059B1868238 51EC7DD9CA1161DE93D5174D66E8382E9BB2FE84E47

7.1 Contribution to thesis goals

This section contributes to goals G3, G4 and G5. We have generated curves Curve224 and Curve272 that satisfy the `SafeCurves` requirements. We used these curves to construct EC cryptosystems, thus demonstrating the potential of our curves in Signal protocol.

8 References

[1] Alejandra Alvarado. An exposition of Schoof's algorithm. https://mathpost.asu.edu/~sjgm/issues/2005_spring/SJGM_alvarado.pdf.

[2] Harald Baier and Johannes Buchmann. Generation Methods of Elliptic Curves. Information-technology Promotion Agency, Japan. https://bit.ly/34lvcQT. 2002.

[3] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards Curves. Cryptology ePrint Archive. https://eprint.iacr.org/2008/013.

[4] Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. How to manipulate curve standards: a white paper for the black hat. Cryptology ePrint Archive. https://eprint.iacr.org/2014/571.

[5] Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography. https://safecurves.cr.yp.to.

[6] Daniel J. Bernstein and Tanja Lange. Two grumpy giants and a baby. Cryptology ePrint Archive. https://eprint.iacr.org/2012/294.

[7] Aaron Blumenfeld. Pollard's Rho algorithm for elliptic curves. https://bit.ly/2PD1C5U. 2015.

[8] Silje Christensen and Simen Johnsrud. Speeding up the Pollard's Rho algorithm. https://bit.ly/2LdzUIM.

[9] Harold Edwards. A normal form for elliptic curves. Bulletin of the American Mathematical Society. https://bit.ly/2NQylSy. 2007.

[10] epic.org. Facebook Privacy. https://epic.org/privacy/facebook/.

[11] Jean-Pierre Flori, Jérôme Plût, Jean-René Reinhard, and Martin Ekerå. "Diversity and transparency for ECC". In: Cryptology ePrint Archive 659 (2015).

[12] Zcash foundation. *What is Jubjub?* `https://z.cash/technology/jubjub/`.

[13] Steven D. Galbraith and Pierrick Gaudr. Recent progress on the elliptic curve discrete logarithm problem. `https://eprint.iacr.org/2015/1022.pdf`.

[14] Mike Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive. `https://eprint.iacr.org/2015/625`.

[15] Ståle Zerener Haugnæss. On the Generation of Strong Elliptic Curves For Cryptographic Applications. Master's Thesis. University of Oslo. `https://bit.ly/2ZHoxRk`. 2015.

[16] José de Jesús Angel Angel and Guillermo Morales-Luna. *Solinas primes of small weight for fixed sizes*. Cryptology ePrint Archive, Report 2010/058. `https://eprint.iacr.org/2010/058`. 2010.

[17] A. Langley, M. Hamburg, and S. Turner. Alternative Elliptic Curve Representations. Internet-Draft. IETF Tools, Mar. 2020.

[18] A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748. RFC Editor, Jan. 2016.

[19] Christof Paar and Jan Pelzl. *Understanding Cryptography*. Springer, 2010.

[20] Bruce Schneier. More on Backdooring (or Not) WhatsApp. `https://www.schneier.com/blog/archives/2019/08/more_on_backdoo.html`.

[21] Berry Schoenmakers and Andrey Sidorenko. Cryptanalysis of the Dual Elliptic Curve Pseudorandom Generator. Cryptology ePrint Archive. `https://eprint.iacr.org/2006/190`. 2006.

[22] Martin Lysoe Sommerseth and Haakon Hoeiland. Pohlig-Hellman Applied in Elliptic Curve Cryptography. `https://bit.ly/2HDLrib`. 2015.

[23] Douglas R. Stinson and Maura B. Paterson. *Cryptography: Theory and Practice, Fourth Edition*. 2019.

[24] Suresh Sundriyal. Counting points on elliptic curves over Z_p Master's Thesis. Rochester Institute of Technology. Accessed from `https://scholarworks.rit.edu/theses/353/`. 2008.

[25] Peter Wozny. Elliptic curve cryptography: Generation and validation of domain parameters in binary Galois Fields. Master's Thesis. Rochester Institute of Technology. Accessed from `https://scholarworks.rit.edu/theses/354/`. 2008.

[26] Gregorio Zanon. No, end-to-end encryption does not prevent Facebook from accessing WhatsApp chats. `https://bit.ly/2JKAMC0`.

9 Appendix 1: Order of the curves

9.1 Order of NIST P-256 using Schoof's algorithm

NIST P-256 are of the equation: $y^2 = x^3 - 3x + b \bmod q$

```
wine schoof.exe -f 2#256-2#224+2#192+2#96-1 -3
4105836372515214212932612978004726840911444101599372555
83525631403946740129 1
P mod 8 = 7
P is 256 bits long
Counting the number of points (NP) on the curve
y^2= x^3 - 3*x + 4105836372515214212932612978004726840
91144410159937255548352563140394674012 91 mod 11579208921
03562487626974469494075735300861434152903141955336313088
67097853951
18 primes used (plus largest prime powers), largest is 61
NP mod 2 =   1
NP mod 3 =   1
NP mod 5 =   4
NP mod 7 =   3
NP mod 11 =   7
NP mod 13 =   5
NP mod 17 =  13
NP mod 19 =  10
NP mod 23 =  10
NP mod 25 =  19
NP mod 27 =   7
NP mod 29 =  23
NP mod 31 =   9
```

```
NP mod 32 =  17
NP mod 37 =  35
NP mod 41 =  15
NP mod 43 =  34
NP mod 47 =   2
NP mod 49 =  38
NP mod 53 =  41
NP mod 59 =  34
NP mod 61 =  30
Releasing 5 Tame and 5 Wild Kangaroos
NP = 1157920892103562487626974469494075735299969552241357603424222
59061068512044369
NP is Prime!
```

The cofactor for this curve is $h = 1$. Hence, the order of the curve is $NP = 1157920892103562487626974469494075735299969552241357603424222$
59061068512044369

9.2 Sec256k1 - Bitcoin curve

```
wine schoof.exe -f 2#256-2#32-2#9-2#8-2#7-2#6-2#4-1 0 7
P mod 8 = 7
P is 256 bits long
Counting the number of points (NP) on the curve
y^2= x^3 + 7 mod 115792089237316195423570985008687907853
269984665640564039457584007908834671663
Warning: j-invariant is 0
18 primes used (plus largest prime powers), largest is 61
NP mod 2 =  1
NP mod 3 =  1
```

```
NP mod 5 =    2
NP mod 7 =    3
NP mod 11 =   3
NP mod 13 =  12
NP mod 17 =   7
NP mod 19 =   8
NP mod 23 =   8
NP mod 25 =  12
NP mod 27 =  25
NP mod 29 =  24
NP mod 31 =   9
NP mod 32 =   1
NP mod 37 =  17
NP mod 41 =   9
NP mod 43 =  29
NP mod 47 =  44
NP mod 49 =  38
NP mod 53 =  35
NP mod 59 =   9
NP mod 61 =   8
Releasing 5 Tame and 5 Wild Kangaroos
.....................................................
NP = 1157920892373161954235709850086879078528375642790749
043826051631415181614943337
NP is Prime!
```

The cofactor for this curve is $h = 1$. Hence, the order of the curve is NP

9.3 Order of Curve25519 using Schoof's algorithm

Weierstrass curve-specific parameters (for Wei25519):

```
wine schoof.exe -f 2#255-19
192986815395526992372618308347813179755449974442734273399095973345732416392365575174666981890890764528907825714081824110372790101231529440083795672935843 6
P mod 8 = 5
P is 255 bits long
Counting the number of points (NP) on the curve
y^2= x^3 + 192986815395526992372618308347813179755449974442734273399095973345732416392365*x - 214429794883918880414020342608681310839388860491926970443439116599983546151357896044618658097711785492504343953926634992332820282019728792003956564819949
18 primes used (plus largest prime powers), largest is 61
NP mod 2 =   0 ***
NP mod 3 =   2
NP mod 5 =   2
NP mod 7 =   6
NP mod 11 =   8
NP mod 13 =   3
NP mod 17 =  13
NP mod 19 =   1
NP mod 23 =  17
NP mod 25 =  12
NP mod 27 =   5
NP mod 29 =  28
NP mod 31 =  30
```

```
NP mod 32 =   8
NP mod 37 =  18
NP mod 41 =  18
NP mod 43 =  41
NP mod 47 =  21
NP mod 49 =  48
NP mod 53 =  24
NP mod 59 =  11
NP mod 61 =   7
Releasing 5 Tame and 5 Wild Kangaroos
.........................
NP = 57896044618658097711785492504343953926856930875039260848015607506283634007912
```

The order of the curve as per the RFC is

$2^{252} + 27742317777372353535851937790883648493$	$7.2370056e + 75$

We can see that order of the curve multiplied by the 8 is equal to the total points on the curve. 8 is a called a cofactor.

578960446186580977117854925043439539268569308750392608480156075062836340079128	$7.2370056e + 75$

9.4 Order of Curve25519 using SageMath

```
sage: ec = EllipticCurve(GF(2**255-19), [0,486662,0,1,0])
sage: ec
Elliptic Curve defined by y^2 = x^3 + 486662*x^2 + x
over finite field of size 57896044618658097711785492504343953926856930875039260848015607506283634007912
```

```
sage: ec.order()
57896044618658097711785492504343953926856930875039260848015607506283634007912
```

9.5 Order of Curve448 using SageMath

```
sage: ec = EllipticCurve(GF(2**448 - 2**224 - 1),
[0,156326,0,1,0])
sage: ec
Elliptic Curve defined by y^2 = x^3 + 156326*x^2 + x
over finite field of size 726838724295606890549323807888004534353641360687318060281490199180584015846158342864783021166769503853241174836366492190950234385991163
```

```
sage: ec.order()
726838724295606890549323807888004534353641360687318060281490199180584015846158342864783021166769503853241174836366492190950234385991163
```

9.6 Order of Curve M-511 using SageMath

```
sage: ec = EllipticCurve(GF(2**511 - 187),
[0,530438,0,1,0])
sage: ec
Elliptic Curve defined by $y^2 = x^3 + 530438*x^2 + x$
over finite field of size 67039039649712985497870124991029
23063739682910296196688861780721860882015036773488400937 14
90834517138450159290932430254268769414059732849732168245030
41861

sage: ec.order()
67039039649712985497870124991029230637396829102961966888617
80721860882015036859286439014235064444070097128474067979591
47989642007020500929968744590353839 2
```

9.7 Time analysis of Schoof's algorithm

We calculate the time required for Schoof's algorithm to count the points on the curve. It is essential to understand long the algorithm takes to run given specific curves.

9.7.1 Curve 25519

Counting the number of points (NP) on the curve:

$y^2 = x^3 + A * x - B$ mod 57896044618658097711785492504343953926634
99233282028201972879200395656481994 9,

where

A =192986815395526992372618308347813179755449974442734273399095973345732416392 36,
B = 214429794883918880414020342608681310839388860491926970443439116599983546 1513

NP = 578960446186580977117854925043439539268569308750392608480156075062836340 07912
Time required by the algorithm to compute the points is : 387.4464828968048 seconds.

9.7.2 NIST P-256

Counting the number of points (NP) on the curve

$y^2 = x^3 + A * x + B$ mod 115792089210356248762697446949407573530086143415290314195533631308867097853951,

where
$A = -3$,
B = 41058363725152142129326129780047268409114441015993725554835256314039467402 91

NP = 115792089210356248762697446949407573529996955224135760342422259061068512044369
NP is Prime!
Time required by the algorithm to compute the points is : 508.2849681377411 seconds.

9.7.3 Sec256k1 - Bitcoin curve

Counting the number of points (NP) on the curve

$y^2 = x^3 + 7 \bmod 115792089237316195423570985008687907853269984665640564039457584007908834671663$

$NP = 115792089237316195423570985008687907852837564279074904382605163141518161494337$ NP is Prime!

Time required by the algorithm to compute the points is : 521.7670800685883 seconds.

9.7.4 Weierstrass Curves

Weierstrass curves that are generated over prime fields P-192, P-224, P-256, P-384, and P-521 which have equation $y^2 = x^3 + ax + b$.

P-192 Only for legacy use.

P-224

p	$2^{224} - 2^{96} + 1$
a	-3
b	18958286285566608000408668544493926415504680968679321075787234672564
h	1
n	26959946667150639794667015087019625940457807714424391721682722368061

P-256

p	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
a	-3
b	41058363725152142129326129780047268409114441015993725554835256314039467401291
h	1
n	1157920892103562487626974469494075735291157920892103562487626974469494075735529

P-384

p	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
a	-3
b	27580193559959705877849011840389048093056905856361568521422758019355995970587784901184038904809305690585636156852142
h	1
n	27580193559959705877849011840389048093056905856361568521422758019355995970587784901184038904809305690585636156852142

P-521

p	$2^{521} - 1$
a	-3
b	1093849038073734274511112390766805569936207598951683748994586394495953116150735016013708737573759623248592132296706313309438452531591012912142327488478985984
h	1
n	6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449

W-25519

p	$2^{255} - 19$
a	19298681539552699237261830834781317975544997444273427339909597334573241639236
b	55751746669818908907645289078257140818241103727901012315294400837956729358436
h	8
n	72370055773322622139731865630429942408571163593799076060019509382854542509 89

W-448

p	$2^{448} - 2^{224} - 1$
a	484559149530404593699549205258669689569094240458212040187660132787074885444487181790930922465784363953392589641229091574035657199637535
b	269199527516891440944194002921483160871719022476784466770922295992819380802492878772739401369880202196329216467349495319191685664513904
h	4
n	181709681073901722637330951972001133588410340171829515070372549795146003961539585716195755291692375963310293709091662304773755859649779

9.7.5 Montgomery Curves

Montgomery curves have equation $By^2 = x^3 + Ax^2 + x$

Curve25519

p	$2^{255} - 19$
A	486662
B	1
h	8
n	7237005577332262213973186563042994240857116359379907606001 950938285454250989

Curve448

p	$2^{448} - 2^{224} - 1$
A	156326
B	1
h	4
n	181709681073901722637330951972001133588410340171829515070372549795146003961539585716195755291692375963310293709091662304773755859649779

9.7.6 Twisted Edwards Curves

Twisted Edwards curves have equation $ax^2 + y^2 = 1 + dx^2y^2$

Edwards25519

p	$2^{255} - 19$
a	-1
d	$-121665/121666$
h	8
n	7237005577332262213973186563042994240857116359379907606001 950938285454250989

E448

p	$2^{448} - 2^{224} - 1$
a	1
d	39082/39081
h	4
n	181709681073901722637330951972001133588410340171829515070372549795146003961539585716195755291692375963310293709091662304773755859649779

Edwards448

p	$2^{448} - 2^{224} - 1$
a	1
d	-39081
h	4
n	181709681073901722637330951972001133588410340171829515070372549795146003961539585716195755291692375963310293709091662304773755859649779

9.7.7 Koblitz Curves

Koblitz curve has equation $y^2 + xy = x^3 + ax^2 + b$

Curve K-163 Only for legacy use.

Curve K-233

$f(z)$	$z^{233} + z^{74} + 1$
a	0
b	1
h	4
n	3450873173395281893717377931138512760570940988862252126328 087024741343

Curve K-283

$f(z)$	$z^{283} + z^{12} + z^7 + z^5 + 1$
a	0
b	1
h	4
n	3885337784451458141838923813647037813284811733793061324295 874997529815829704422603873

Curve K-409

$f(z)$	$z^{409} + z^8 + 1$
a	0
b	1
h	4
n	330527984395124299475957654016385519914202341482140609642324395022880711289249191050673258457777458014096366590617731358671

Curve K-571

$f(z)$	$z^{571} + z^{10} + z^5 + z^2 + 1$
a	0
b	1
h	4
n	1932268761508629172347675945465993672149463664853217499328617625725759571144780212268133978522706711834706712800825351461273674974066617311929682421617092503555733685276673

Curve B-163 Only for legacy use.

Curve B-233

$f(z)$	$z^{233} + z^{74} + 1$
a	1
h	2
n	6901746346790563787434755862277025555839812737345013555379383634485463

Curve B-283

$f(z)$	$z^{283} + z^{12} + z^7 + z^5 + 1$
a	1
h	2
n	77706755689029162836778476272940756265696259243769048891091965267700442777873786928 71

Curve B-409

$f(z)$	$z^{409} + z^8 + 1$
a	1
h	2
n	66105596879024859895191530803277103982840468296428121928464879830415777482737480520814372376217911096597986728836656752677 1

Curve B-571

$f(z)$	$z^{571} + z^{10} + z^5 + z^2 + 1$
a	1
h	2
n	38645375230172583446953518909319873442989273297064349986572352514515191422895604245361439993894157730831338811219269444862468724628168130702345282883033324113931911052857031

10 Appendix 2: Code

All the code used in this thesis can be found here: `https://github.com/Tanay-D/Masters-Thesis`

10.1 Montgomery curves

We have written the code for Montgomery curve point arithmetic and montgomery ladder in C++ using GMP library. To run the code, we have to make a class object for montgomery curve and then invoke the necessary class methods.

Montgomery point addition

```
/**
 * @brief addition of two points on a montgomery curve
 * @param R
 * @param P
 * @param Q
 * @return void
 *
 * Computes R = P+Q
 * GMP Implementation of Montgomery curve point addition formula.
 */
void MontCurve::add(Point& R,const Point& P,const Point& Q,MontCurve E){


  mpz_class tempneg;
```

```
15    tempneg = -Q.Y;
16
      ↪  mpz_mod(tempneg.get_mpz_t(),tempneg.get_mpz_t(),E.p.get_mpz_t());
17
18
19
20    //R = P+Q
21    //assuming all the conditions are satisfied
22
23    //If P = Q
24    if(P == Q){
25        E.ecdouble(R,P,E);
26      }
27
28    // P != Q or -Q
29
30
31
32    else{
33    mpz_class f = E.p;
34    mpz_class one = 1;
35    mpz_class xtop,ytop,xtemp,ytemp,lambda;
36    mpz_class neg = -P.Y;
37
38
39    if(Q.X == 0 && Q.Y == 1){
40        R = P;
41        return;
42      }
43
```

```
44      else if(P.X == 0 && P.Y == 1){
45          R = Q;
46          return;
47        }
48
49      //////////////     Step 1  ///////////////////
50      //computing lambda =   = (x2 - x1)/(y2 - y1)
51      //           (yQ - yP )/(xQ - xP )
52      xtemp = (Q.X - P.X);
53      mpz_mod(xtemp.get_mpz_t(),xtemp.get_mpz_t(),f.get_mpz_t());
        ↪  //mod
54
        ↪  mpz_invert(xtemp.get_mpz_t(),xtemp.get_mpz_t(),f.get_mpz_t());
55
56
57      ytemp = (Q.Y - P.Y);
58      mpz_mod(ytemp.get_mpz_t(),ytemp.get_mpz_t(),f.get_mpz_t());
59
60
61      //multiply the top with inverse and store it in lambda
62      lambda = xtemp*ytemp;
63      mpz_mod(lambda.get_mpz_t(),lambda.get_mpz_t(),f.get_mpz_t());
64
65      // add debug on settings
66      //std::cout << "lambda : " << lambda.get_str() <<
        ↪  std::endl;
67
68      //////////////     Step 2  ///////////////////
69      //x3 = B ^2 - A - x1 - x2
70      R.X = E.B *lambda * lambda - E.A - P.X - Q.X;
```

```
71    mpz_mod(R.X.get_mpz_t(),R.X.get_mpz_t(),f.get_mpz_t());
72    //std::cout << "x3 : " << R.X.get_str() << std::endl;
73
74
75    //////////////     Step 3  ///////////////////
76    //y3 = (xP - x3) -    yP
77    R.Y = lambda*(P.X - R.X) - P.Y;
78    mpz_mod(R.Y.get_mpz_t(),R.Y.get_mpz_t(),f.get_mpz_t());
79
80      }
81    }
```

Montgomery ladder

```
1  /**
2   * @brief point multiplication using montgomery ladder method
3   * @param R
4   * @param P
5   * @param x
6   * @param M
7   * @return void
8   *
9   * Computes R = x[P]
10  * GMP Implementation of Montgomery ladder.
11  */
12 void MontCurve::Mladder(Point& R, const Point& P,const
    ↪  mpz_class x, MontCurve& M){
13
```

```
14    Point R0("0","1");
15    Point R1 = P;
16
17    Point temp0,temp1;
18    int i;
19    string value_str = x.get_str(2);
20    int arr[value_str.length()];
21
22
23    for (i = 0; i < value_str.length(); i++) {
24        arr[i] = value_str[i] - '0';
25    }
26
27    for (i = 0 ; i <= value_str.length()-1  ; i++){
28
29        if(arr[i] == 0){
30
31            M.add(temp1,R1,R0,M);
32            M.ecdouble(temp0,R0,M);
33
34            R1 = temp1;
35            R0 = temp0;
36
37          }
38        else{
39            M.add(temp0,R0,R1,M);
40            M.ecdouble(temp1,R1,M);
41
42            R0 = temp0;
43            R1 = temp1;
```

```
44          }
45
46      }
47
48
49      R = R0;
50  }
```

Montgomery point doubling

```
1   /**
2    * @brief point multiplication using montgomery ladder method
3    * @param R
4    * @param P
5    * @param E
6    * @return void
7    *
8    * Computes R = x[P]
9    * GMP Implementation of Montgomery point doubling.
10   */
11
12  void MontCurve::ecdouble(Point& R,const Point& P,MontCurve E){
13
14    mpz_class f = E.p;
15    mpz_class one = 1;
16    mpz_class xtop,ytop,xtemp,ytemp,lambda;
17
18
```

```
19
20
21      if(P.X == 0 && P.Y == 1){
22          R = P;
23          return;
24        }
25
26
27
28      //////////////     Step 1  ///////////////////
29      //computing lambda =   = (3 x1^2 + 2Ax1 + 1)/2By1
30
31      xtemp = 3*P.X*P.X + 2*E.A*P.X + 1;
32      mpz_mod(xtemp.get_mpz_t(),xtemp.get_mpz_t(),f.get_mpz_t());
        ↪  //mod
33
34      ytemp = 2*E.B*P.Y;
35
        ↪  mpz_invert(ytemp.get_mpz_t(),ytemp.get_mpz_t(),f.get_mpz_t());
36
37      //multiply the top with inverse and store it in lambda
38      lambda = xtemp*ytemp;
39      mpz_mod(lambda.get_mpz_t(),lambda.get_mpz_t(),f.get_mpz_t());
40
41      //////////////     Step 2  ///////////////////
42      //computing x3 + 2x1 = B 2 - A
43
44      R.X = (E.B *lambda*lambda)-E.A-(2*P.X);
45      mpz_mod(R.X.get_mpz_t(),R.X.get_mpz_t(),f.get_mpz_t());
46
```

```
47
48    //////////////     Step 3  ///////////////////
49    // computing y3 = y3 + y1 = (x1 - x3)
50
51    R.Y = lambda*(P.X - R.X) - P.Y;
52    mpz_mod(R.Y.get_mpz_t(),R.Y.get_mpz_t(),f.get_mpz_t());
53
54
55  }
```

10.2 Twisted Edwards curves

We have written the code for twisted edwards curve point arithmetic in C++ using GMP library. To run the code, we have to make a class object for edwards curve and then invoke the necessary class methods.

Twisted edwards curve point addition

```
1   /**
2    * @brief add
3    * @param R
4    * @param P
5    * @param Q
6    * @param d
7    * @param eda
8    * @return void
9    *
10   * Computes R = P+Q
11   * GMP Implmentation of twisted Edwards curve addition
    ↪  formula.
```

```
12  */
13
14  void ECurve::add(Point& R,const Point& P,const Point& Q,ECurve
    ↪  E){
15
16      mpz_class f = E.p;
17      mpz_class one = 1;
18      mpz_class d_ = E.d;
19      mpz_class a_ = E.a;
20      mpz_class xtop,ytop,xtemp,ytemp;
21
22
23      //(x1+y1) + (x2+y1)
24      xtop = (P.X * Q.Y) + (Q.X * P.Y);
25      mpz_mod(xtop.get_mpz_t(),xtop.get_mpz_t(),f.get_mpz_t());
26
27      //1 + d*x1*x2*y1*y2
28      xtemp = one + d_*P.X*P.Y*Q.X*Q.Y;
29      mpz_mod(xtemp.get_mpz_t(),xtemp.get_mpz_t(),f.get_mpz_t());
30
        ↪  mpz_invert(xtemp.get_mpz_t(),xtemp.get_mpz_t(),f.get_mpz_t());
31
32      //multiply the top with inverse and store it in xtop
33      xtop = xtop*xtemp;
34      mpz_mod(xtop.get_mpz_t(),xtop.get_mpz_t(),f.get_mpz_t());
35
36
37      ytop = (P.Y * Q.Y) - (a_*P.X*Q.X);
38      mpz_mod(ytop.get_mpz_t(),ytop.get_mpz_t(),f.get_mpz_t());
39
```

```
40      //d-x1x2y1y2
41      ytemp = one - d_*P.X*P.Y*Q.X*Q.Y;
42      mpz_mod(ytemp.get_mpz_t(),ytemp.get_mpz_t(),f.get_mpz_t());
43
        ↪  mpz_invert(ytemp.get_mpz_t(),ytemp.get_mpz_t(),f.get_mpz_t());
44
45      //multiply the top with inverse and store it in xtop
46      ytop = ytop*ytemp;
47      mpz_mod(ytop.get_mpz_t(),ytop.get_mpz_t(),f.get_mpz_t());
48
49      swap(R.X,xtop);
50      swap(R.Y,ytop);
51
52      }
53  }
```

Twisted edwards curve point doubling

```
1   /**
2    * @brief add
3    * @param R
4    * @param P
5    * @param E
6    * @return void
7    *
8    * Computes R = 2*P
9    * GMP Implmentation of twisted Edwards curve doubling
    ↪  formula
```

```
 */
void ECurve::ecdouble(Point& R,const Point& P,ECurve E){
     //R = P+P
     ECurve::add(R,P,P,E);
}
```

Twisted edwards curve scalar multiplication

```
/**
 * @brief add
 * @param R
 * @param P
 * @param x
 * @param E
 * @return void
 *
 * Computes R = x*P
 * GMP Implmentation of double and add scalar multiplication
 ↪  algorithm.
 */


//core sclar multiplication
Point& ECurve::scalarmult(Point &R, const Point &P, mpz_class
 ↪  x, ECurve &E){

  if(x == 0){
      R = identity_element;
```

```
19        return R;
20      }
21
22    R = ECurve::scalarmult(R,P,x/2,E);
23
24    ECurve::add(R,R,R,E);
25
26    if (x % 2 == 1){
27      ECurve::add(R,R,P,E);
28      }
29
30    return R;
31  }
32
33  //Helper function
34  void ECurve::scalar_mult(Point& R, const Point& P,mpz_class x,
    ↪  ECurve& E){
35       R = ECurve::scalarmult(R,P,x,E);
36
37  }
```