Mechanics of MobileCoin: First Edition

exploring the foundations of a private digital currency

December 8, 2020, Preview (2/10) v0.0.16

koe[1,2]

DRAFT INFORMATION: This is just a draft, and may not always be available wherever it is currently hosted. The final version will be published at https://github.com/mobilecoinfoundation.

[1] ukoe@protonmail.com

[2] Author 'koe' worked on this document as part of a private contract with MobileCoin, Inc.

Abstract

Cryptography. It may seem like only mathematicians and computer scientists have access to this obscure, esoteric, powerful, elegant topic. In fact, many kinds of cryptography are simple enough that anyone can learn their fundamental concepts.

It is common knowledge that cryptography is used to secure communications, whether they be coded letters or private digital interactions. Another application is in so-called cryptocurrencies. These digital moneys use cryptography to assign and transfer ownership of funds. To ensure that no piece of money can be duplicated or created at will, cryptocurrencies usually rely on 'blockchains', which are public, distributed ledgers containing records of currency transactions that can be verified by third parties [52].

It might seem at first glance that transactions need to be sent and stored in plain text format to make them publicly verifiable. In truth, it is possible to conceal a transaction's participants, as well as the amounts involved, using cryptographic tools that nevertheless allow transactions to be verified and agreed upon by observers [66]. This is exemplified in the cryptocurrency MobileCoin.

We endeavor here to teach anyone who knows basic algebra and simple computer science concepts like the 'bit representation' of a number not only how MobileCoin works at a deep and comprehensive level, but also how useful and beautiful cryptography can be.

For our experienced readers: MobileCoin is a standard one-dimensional distributed acyclic graph (DAG) cryptocurrency blockchain [52], where blocks are consensuated with a Byzantine Federated Agreement protocol [48], transactions are validated in SGX secure enclaves [20] and are based on elliptic curve cryptography using the Ristretto abstraction [31] on curve Ed25519 [14], transaction inputs are shown to exist in the blockchain with Merkle proofs of membership [49] and are signed with Schnorr-style multilayered linkable spontaneous anonymous group signatures (MLSAG) [54], and output amounts (communicated to recipients via ECDH [24]) are concealed with Pedersen commitments [46] and proven in a legitimate range with Bulletproofs [18].

Contents

CHAPTER 1

Introduction

In the digital realm it is often trivial to make endless copies of information, with equally endless alterations. For a currency to exist digitally and be widely adopted, its users must believe its supply is strictly limited. A money recipient must trust they are not receiving counterfeit coins, or coins that have already been sent to someone else. To accomplish these goals without requiring the collaboration of any third party like a central authority, the currency's supply and complete transaction history must be publicly verifiable.

We can use cryptographic tools to allow data registered in an easily accessible database — the blockchain — to be virtually immutable and unforgeable, with legitimacy that cannot be disputed by any party.

Cryptocurrencies typically store transactions in the blockchain, which acts as a public ledger[1] of all the currency operations. Most cryptocurrencies store transactions in clear text, to facilitate verification of transactions by the community of users.

Clearly, an open blockchain defies any basic understanding of privacy or fungibility[2], since it

[1] In this context ledger just means a record of all currency creation and exchange events. Specifically, how much money was transferred in each event and to whom.

[2] "**Fungible** means capable of mutual substitution in use or satisfaction of a contract. A commodity or service whose individual units are so similar that one unit of the same grade or quality is considered interchangeable with any other unit of the same grade or quality. Examples: tin, grain, coal, sugar, money, etc." [7] In an open blockchain such as Bitcoin, the coins owned by Alice can be differentiated from those owned by Bob based on the 'transaction history' of those coins. If Alice's transaction history includes transactions related to supposedly nefarious actors, then her coins might be 'tainted' [51], and hence less valuable than Bob's (even if they own the same amount of coins). Reputable figures claim that newly minted Bitcoins trade at a premium over used coins, since they don't have a history [56].

literally *publicizes* the complete transaction histories of its users.

To address the lack of privacy, users of cryptocurrencies such as Bitcoin can obfuscate transactions by using temporary intermediate addresses [53]. However, with appropriate tools it is possible to analyze flows and to a large extent link true senders with receivers [61, 16, 55, 19].

In contrast, the cryptocurrency MobileCoin attempts to tackle the issue of privacy by storing only single-use addresses for ownership of funds in the blockchain, authenticating the dispersal of funds in each transaction with ring signatures, and verifying transactions within black-box 'secure enclaves' that discard extraneous information after verification. With these methods there are no known effective ways to link receivers or trace the origin of funds.[3]

Additionally, transaction amounts in the MobileCoin blockchain are concealed behind cryptographic constructions, rendering currency flows opaque even in the case of secure enclave failures.

The result is a cryptocurrency with a high level of privacy and fungibility.

1.1 Objectives

MobileCoin is a new cryptocurrency employing a novel combination of techniques. Many of its aspects are either backed by technical documents missing key details pertinent to MobileCoin, or by non-peer-reviewed papers that are incomplete or contain errors.[4] Other aspects can only be understood by examining the source code and source code documentation (comments and READMEs) directly.

Moreover, for those without a background in mathematics, learning the basics of elliptic curve cryptography, which MobileCoin uses extensively, can be a haphazard and frustrating endeavor.

We intend to address this situation by introducing the fundamental concepts necessary to understand elliptic curve cryptography, reviewing algorithms and cryptographic schemes, and collecting in-depth information about MobileCoin's inner workings.

To provide the best experience for our readers, we have taken care to build a constructive, step-by-step description of the MobileCoin cryptocurrency.

In the first edition of this report we have centered our attention on version 1 of the MobileCoin protocol[5], corresponding to version 1.0.0 of the MobileCoin software suite. All transaction and blockchain-related mechanisms described here belong to those versions.[6]

[3] Depending on the behavior of users, if an attacker manages to break into secure enclaves there may be cases where transactions can be analyzed to some extent. For an example see this article: [27].

[4] Seguias has created the excellent Monero Building Blocks series [60], which contains a thorough treatment of the cryptographic security proofs used to justify Monero's signature schemes. Seguias's series is focused on v7 of the Monero protocol, however much of what he discusses is applicable to MobileCoin, which uses many of the same building blocks as Monero.

[5] The 'protocol' is the set of rules that each new block is tested against before it can be added to the blockchain. This set of rules includes the 'transaction protocol' (currently version 1, which we call `TXTYPE_RCT_1` for clarity), which are general rules pertaining to how a transaction is constructed.

[6] The MobileCoin codebase's integrity and reliability is predicated on assuming enough people have reviewed

1.2 Readership

We anticipate many readers will encounter this report with little to no understanding of discrete mathematics, algebraic structures, cryptography[7], or blockchains. We have tried to be thorough enough that laypeople with a diversity of backgrounds may learn about MobileCoin without needing external research.

We have purposefully omitted, or delegated to footnotes, some mathematical technicalities, when they would be in the way of clarity. We have also omitted concrete implementation details where we thought they were not essential. Our objective has been to present the subject half-way between mathematical cryptography and computer programming, aiming at completeness and conceptual clarity.[8]

1.3 Origins of the MobileCoin cryptocurrency

MobileCoin's whitepaper was released in November 2017 by Joshua Goldbard and Moxie Marlinspike. According to the paper, their motivation for the project was to "develop a fast, private, and easy-to-use cryptocurrency that can be deployed in resource constrained environments to users who aren't equipped to reliably maintain secret keys over a long period of time, all without giving up control of funds to a payment processing service." [30]

1.4 Outline

Since MobileCoin is a very new cryptocurrency, we will not go into detail on hypothetical second-layer extensions and applications of the core protocol in this edition. Suffice it to say for now that topics like multisignatures, transaction proofs, and escrowed marketplaces are just as feasible for MobileCoin as they are for any well-designed progeny of the CryptoNote protocol.[9]

---

it to catch most or all significant errors. We hope that readers will not take our explanations for granted, and verify for themselves the code does what it's supposed to. If it does not, we hope you will make a responsible disclosure (by emailing `security@mobilecoin.foundation`) for major problems, or Github pull request (`https://github.com/mobilecoinfoundation/mobilecoin`) for minor issues.

[7] An extensive textbook on applied cryptography can be found here: [17].

[8] Some footnotes, especially in chapters related to the protocol, spoil future chapters or sections. These are intended to make more sense on a second read-through, since they usually involve specific implementation details that are only useful to those who have a grasp of how MobileCoin works.

[9] Monero, initially known as BitMonero, was created in April 2014 as a derivative of the proof-of-concept currency CryptoNote [64]. Subsequent changes to Monero's transaction type retained parts of the original CryptoNote design (specifically, one-time addresses, ring signatures of one form or another, and key images). This means CryptoNote is in some sense an ancestor of MobileCoin, whose transaction scheme is inspired by Monero's `RCTTypeBulletproof2` transaction type.

1.4.1 Essentials

In our quest for comprehensiveness, we have chosen to present all the basic elements of cryptography needed to understand the complexities of MobileCoin, and their mathematical antecedents. In Chapter 2 we develop essential aspects of elliptic curve cryptography.

Chapter 3 expands on the Schnorr signature scheme introduced in the prior chapter, and outlines the ring signature algorithms that will be applied to achieve confidential transactions. Chapter 4 explores how MobileCoin uses addresses to control ownership of funds, and the different kinds of addresses.

In Chapter 5 we introduce the cryptographic mechanisms used to conceal amounts. Chapter 6 is dedicated to membership proofs, which are used to prove MobileCoin transactions spend funds that exist in the blockchain. With all the components in place, we explain the transaction scheme used by MobileCoin in Chapter 7.

We shed light on secure enclaves in Chapter 8, and the MobileCoin blockchain is unfolded in Chapter 9.

1.4.2 Extensions

A cryptocurrency is more than just its protocol. As part of MobileCoin's original design, a 'service-layer' technology known as Fog was developed. Discussed in Chapter 10, Fog is a service that searches the blockchain and identifies transaction outputs owned by its users. This allows users to avoid scanning the blockchain themselves, which is time-consuming and resource-intensive (a prohibitive burden for e.g. mobile devices). Importantly, the service operator is not able to learn anything about those outputs.

1.4.3 Additional content

Appendix A explains the structure of a sample transaction from the blockchain. Appendix B explains the structure of blocks in MobileCoin's blockchain. Finally, Appendix C brings our report to a close by explaining the structure of MobileCoin's origin (a.k.a. genesis) block. These provide a connection between the theoretical elements described in earlier sections with their real-life implementation.

We use margin notes to indicate where MobileCoin implementation details can be found in the source code.[10] There is usually a file path, such as transaction/std/src/transaction_builder.rs, and a function, such as `create_output()`. Note: '-' indicates split text, such as Ristretto- Point → RistrettoPoint, and we neglect namespace qualifiers (e.g. `TransactionBuilder::`) in most cases.

Isn't this useful?

[10] Our margin notes are accurate for version 1.0.0 of the MobileCoin software suite, but may gradually become inaccurate as the codebase is constantly changing. However, the code is stored in a git repository (https://github.com/mobilecoinfoundation/mobilecoin), so a complete history of changes is available.

Some code references are to third-party libraries, which we tagged with square brackets. These include

- [dalek25519]: the `dalek-cryptography curve25519-dalek` library [21]
- [dalekBP]: the `dalek-cryptography bulletproofs` library [22]
- [blake2]: the Rust implementation [23] of hashing algorithm Blake2 [11]

To shorten their length, margin notes related to the transaction protocol use the tag [MC-tx], which stands for the directory path 'transaction/core/'.

1.5 Disclaimer

All signature schemes, applications of elliptic curves, and implementation details should be considered descriptive only. Readers considering serious practical applications (as opposed to a hobbyist's explorations) should consult primary sources and technical specifications (which we have cited where possible). Signature schemes need well-vetted security proofs, and implementation details can be found in the source code. In particular, as a common saying among cryptographers and security engineers goes, "don't roll your own crypto". Code implementing cryptographic primitives should be well-reviewed by experts and have a long history of dependable performance.[11]

1.6 History of 'Mechanics of MobileCoin'

'Mechanics of MobileCoin: First Edition' is an adaptation of the public domain 'Zero to Monero: Second Edition', published in April 2020 [43]. 'Zero to Monero' itself (its first edition was published in June 2018 [10]) is an expansion of Kurt Alonso's master's thesis, 'Monero - Privacy in the Blockchain' [9], published in May 2018.

There are several notable differences between 'Mechanics of MobileCoin: First Edition' and 'Zero to Monero: Second Edition'.

- Parts of the core content were improved, such as an updated group theory section (Chapter 2) and better narrative flow in the chapter on transactions (Chapter 7).
- Transaction details specific to MobileCoin replaced details specific to Monero (for example, the addition of membership proofs in Chapter 6, which serve a purpose similar to Monero's output offsets).
- MobileCoin's consensus model (based in part on the Stellar Consensus Protocol [48]) replaced Monero's more standard mining-based (Nakamoto [52]) consensus mechanism (Chapter 9).
- MobileCoin uses secure enclaves extensively (Chapter 8). They have an important role in MobileCoin's privacy model and are essential to the novel Fog technology (Chapter 10).

[11] Cryptographic primitives are the building blocks of cryptographic algorithms. For example, in elliptic curve cryptography the primitives include point addition and scalar multiplication on that curve (see Chapter 2).

1.7 Acknowledgements

This report, like 'Zero to Monero' before it, would not exist without Alonso's original master's thesis [9], so to him I (koe) owe a great debt of gratitude. Robb Walters got me involved with MobileCoin, and is without doubt a force for good in this crazy world. All I can feel is admiration for the team at MobileCoin, who have designed and built something that goes far beyond what anyone could reasonably hope for in a cryptocurrency. Finally, it is hard to express in words how incredible the legacy of modern technology is. MobileCoin would truly be impossible without the prior research and work of countless people, only a tiny subset of whom can be found in this document's bibliography.

CHAPTER 2

Basic Concepts

2.1 A few words about notation

A focal objective of this report was to collect, review, correct, and homogenize all existing information concerning the inner workings of the MobileCoin cryptocurrency, and, at the same time, supply all the necessary details to present the material in a constructive and single-threaded manner.

An important instrument to achieve this was to settle for a number of notational conventions. Among others, we have used:

- lower case letters to denote simple values, integers, strings, bit representations, etc.,
- upper case letters to denote curve points and complicated constructs.

For items with a special meaning, we have tried to use as much as possible the same symbols throughout the document. For instance, a curve generator is always denoted by G, its order is l, private/public keys are denoted whenever possible by k/K respectively, etc.

Beyond that, we have aimed at being *conceptual* in our presentation of algorithms and schemes. A reader with a computer science background may feel we have neglected questions like the bit representation of items, or, in some cases, how to carry out concrete operations. Moreover, students of mathematics may find we disregarded explanations of abstract algebra.

However, we don't see this as a loss. A simple object such as an integer or a string can always be represented by a bit string. So-called 'endianness' is rarely relevant, and is mostly a matter of convention for our algorithms.[1]

Elliptic curve points are normally denoted by pairs (x, y), and can therefore be represented with two integers. However, in the world of cryptography it is common to apply *point compression* techniques that allow a point to be represented using only the space of one coordinate. For our conceptual approach it is often accessory whether point compression is used or not, but most of the time it is implicitly assumed.

We have also used cryptographic hash functions freely without specifying any concrete algorithms. In the case of MobileCoin it will typically be BLAKE2b[2], but if not explicitly mentioned then it is not important to the theory. [blake2] src/`blake2b.rs`

A cryptographic hash function (henceforth simply 'hash function', or 'hash') takes in some message $\mathfrak{m}$ of arbitrary length and returns a hash h (or 'message digest') of fixed length, with each possible output equiprobable for a given input. Cryptographic hash functions are difficult to reverse (called preimage resistance), have an interesting feature known as the *large avalanche effect* that can cause very similar messages to produce very dissimilar hashes, and it is hard to find two messages with the same message digest.

Hash functions will be applied to integers, strings, curve points, or combinations of these objects. These occurrences should be interpreted as hashes of bit representations, or the concatenation of such representations. Depending on context, the result of a hash will be numeric, a bit string, or even a curve point. Further details in this respect will be given as needed.

2.2 Modular arithmetic

Most modern cryptography begins with modular arithmetic, which in turn begins with the modulus operation (denoted 'mod'). We only care about the positive modulus, which always returns a positive integer.

The positive modulus is similar to the 'remainder' after dividing two numbers, e.g. c the 'remainder' of a/n. Let's imagine a number line. To calculate $c = a \pmod{n}$, we stand at point a, then walk

[1] In computer memory, each byte is stored in its own address (an address is akin to a numbered slot, which a byte can be stored in). A given 'word' or variable is referenced by the lowest address of its bytes. If variable x has 4 bytes, stored in addresses 10-13, address 10 is used to find x. The way bytes of x are organized in its set of addresses depends on *endianness*, although each individual byte is always and everywhere stored the same way within its address. Basically, which end of x is stored in the reference address? It could be the *big end* or *little end*. Given x = 0x12345678 (hexadecimal; 2 hexadecimal digits occupy 1 byte e.g. 8 binary digits a.k.a. bits), and an array of addresses {10, 11, 12, 13}, the big endian encoding of x is {12, 34, 56, 78} and the little endian encoding is {78, 56, 34, 12}. [41]

[2] The BLAKE2 hashing algorithm is a successor to the NIST standard *SHA-3* [8] finalist BLAKE. The BLAKE2b variant is optimized for 64-bit platforms. [12]

toward zero with each step $= n$ until we reach an integer ≥ 0 and $< n$. That is c. For example, 4 (modulo 3) = 1, $-5 \pmod 4 = 3$, and so on.

Formally, the positive modulus is here defined for $c = a \pmod n$ as $a = nx + c$, where $0 \leq c < n$ and x is a signed integer that gets discarded (n is a positive non-zero integer).

Note that, if $a \leq n$, $-a \pmod n$ is the same as $n - a$.

2.2.1 Modular addition and multiplication

In computer science it is important to avoid large numbers when doing modular arithmetic. For example, if we have $29 + 87 \pmod{99}$ and we aren't allowed variables with three or more digits (such as $116 = 29 + 87$), then we can't compute $116 \pmod{99} = 17$ directly.

To perform $c = a + b \pmod n$, where a and b are each less than the modulus n, we can do this:

- Compute $x = n - a$. If $x > b$ then $c = a + b$, otherwise $c = b - x$.

We can use modular addition to achieve modular multiplication ($a * b \pmod n = c$) with an algorithm called 'double-and-add'. Let us demonstrate by example. Say we want to do $7 * 8 \pmod 9 = 2$. It is the same as

$$7 * 8 = 8 + 8 + 8 + 8 + 8 + 8 + 8 \pmod 9$$

Now break this into groups of two:

$$(8 + 8) + (8 + 8) + (8 + 8) + 8$$

And again, by groups of two:

$$[(8 + 8) + (8 + 8)] + (8 + 8) + 8$$

The total number of + point operations falls from 6 to 4 because we only need to find (8+8) once.[3]

Double-and-add is implemented by converting the first number (the 'multiplicand' a) to binary (in our example, $7 \to [0111]$), then going through the binary array and doubling and adding. Essentially, we are converting $7 * 8 \pmod 9$ into

$$\begin{aligned} &1 * 2^0 * 8 + 1 * 2^1 * 8 + 1 * 2^2 * 8 + 0 * 2^3 * 8 \\ &= 8 + 16 + 32 + 0 * 64 \end{aligned}$$

Let's make an array $A = [0111]$ and index it 3,2,1,0.[4] A[0] = 1 is the first element of A and is the least significant bit. We set a result variable to be initially $r = 0$, and set a sum variable to be initially $s = 8$ (more generally, we start with $s = b$). We follow this algorithm:

1. Iterate through: $i = (0, ..., A_{size} - 1)$

[3] The effect of double-and-add becomes apparent with large numbers. For example, with $2^{15} * 2^{30}$ straight addition would require about 2^{15} + operations, while double-and-add only requires 15!

[4] This is known as 'LSB 0' numbering, since the least significant bit has index 0. We will use 'LSB 0' for the rest of this chapter. The point here is clarity, not accurate conventions.

   (a) If A[i] $\stackrel{?}{=}$ 1, then $r = r + s \pmod n$.
   (b) Compute $s = s + s \pmod n$.

2. Use the final r: $c = r$.

In our example $7 * 8 \pmod 9$, this sequence appears:

1. $i = 0$
   (a) A[0] = 1, so $r = 0 + 8 \pmod 9 = 8$
   (b) $s = 8 + 8 \pmod 9 = 7$
2. $i = 1$
   (a) A[1] = 1, so $r = 8 + 7 \pmod 9 = 6$
   (b) $s = 7 + 7 \pmod 9 = 5$
3. $i = 2$
   (a) A[2] = 1, so $r = 6 + 5 \pmod 9 = 2$
   (b) $s = 5 + 5 \pmod 9 = 1$
4. $i = 3$
   (a) A[3] = 0, so r stays the same
   (b) $s = 1 + 1 \pmod 9 = 2$
5. $r = 2$ is the result.

2.2.2 Modular exponentiation

Clearly $8^7 \pmod 9 = 8 * 8 * 8 * 8 * 8 * 8 * 8 \pmod 9$. Just like double-and-add, we can do 'square-and-multiply'. For $a^e \pmod n$:

1. Define $e_{scalar} \rightarrow e_{binary}$; $A = [e_{binary}]$; $r = 1$; $m = a$
2. Iterate through: $i = (0, ..., A_{size} - 1)$
   (a) If A[i] $\stackrel{?}{=}$ 1, then $r = r * m \pmod n$.
   (b) Compute $m = m * m \pmod n$.
3. Use the final r as result.

2.2.3 Modular multiplicative inverse

Sometimes we need $1/a \pmod n$, or in other words $a^{-1} \pmod n$. The inverse of something times itself is by definition one (identity). Imagine $0.25 = 1/4$, and then $0.25 * 4 = 1$.

In modular arithmetic, for $c = a^{-1} \pmod n$, $ac \equiv 1 \pmod n$ for $0 \leq c < n$ and for a and n relatively prime [62].[5] Relatively prime means they don't share any divisors except 1 (the fraction a/n can't be reduced/simplified).

We can use square-and-multiply to compute the modular multiplicative inverse when n is a prime number because of *Fermat's little theorem* [4]:

$$\begin{aligned} a^{n-1} &\equiv 1 \pmod n \\ a * a^{n-2} &\equiv 1 \pmod n \\ c \equiv a^{n-2} &\equiv a^{-1} \pmod n \end{aligned}$$

More generally (and more rapidly), the so-called 'extended Euclidean algorithm' [3] can also find modular inverses.

2.2.4 Modular equations

Suppose we have an equation $c = 3 * 4 * 5 \pmod 9$. Computing this is straightforward. Given some operation $\circ$ (for example, $\circ = *$) between two expressions A and B:

$$(A \circ B) \pmod n = [A \pmod n] \circ [B \pmod n] \pmod n$$

In our example, we set $A = 3 * 4$, $B = 5$, and $n = 9$:

$$\begin{aligned} (3 * 4 * 5) \pmod 9 &= [3 * 4 \pmod 9] * [5 \pmod 9] \pmod 9 \\ &= [3] * [5] \pmod 9 \\ c &= 6 \end{aligned}$$

Now we have a way to do modular subtraction (which, as we will see, is not a standalone operation defined for finite fields).

$$\begin{aligned} A - B \pmod n &\rightarrow A + (-B) \pmod n \\ &\rightarrow [A \pmod n] + [-B \pmod n] \pmod n \end{aligned}$$

The same principle would apply to something like $x = (a - b * c * d)^{-1}(e * f + g^h) \pmod n$.[6]

[5] In the equation $a \equiv b \pmod n$, a is *congruent* to $b \pmod n$, which just means $a \pmod n = b \pmod n$.

[6] The modulus of large numbers can exploit modular equations. It turns out $254 \pmod{13} \equiv 2*10*10+5*10+4 \equiv (((2) * 10 + 5) * 10 + 4) \pmod{13}$. An algorithm for $a \pmod n$ when $a > n$ is:

1. Define $A \rightarrow [a_{decimal}]$; $r = 0$
2. For $i = A_{size} - 1, ..., 0$
   (a) $r = (r * 10 + A[i]) \pmod n$
3. Use the final r as result.

2.3 Elliptic curve cryptography

2.3.1 What are elliptic curves?

A finite field $\mathbb{F}_q$, where q is a prime number greater than 3, is the field formed by the set $\{0, 1, 2, ..., q-1\}$. Addition and multiplication $(+, \cdot)$ and negation $(-)$ are calculated $\pmod q$.

[dalek25519] src/backend/serial/[u32|u64]/field.rs

"Calculated $\pmod q$" means $\pmod q$ is performed on any instance of an arithmetic operation between two field elements, or negation of a single field element. For example, given a prime field $\mathbb{F}_p$ with $p = 29$, $17 + 20 = 8$ because $37 \pmod{29} = 8$. Also, $-13 = -13 \pmod{29} = 16$.

Typically, an elliptic curve is defined as the set of all points with coordinates (x, y) satisfying a *Weierstraß* equation [37] (for a given (a, b) pair):[7]

$$y^2 = x^3 + ax + b \quad \text{where} \quad a, b, x, y \in \mathbb{F}_q$$

The cryptocurrency MobileCoin uses a special curve belonging to the category of so-called *twisted Edwards* curves [13], which are commonly expressed as (for a given (a, d) pair):

$$ax^2 + y^2 = 1 + dx^2y^2 \quad \text{where} \quad a, d, x, y \in \mathbb{F}_q$$

In what follows we will prefer this second form. The advantage it offers over the previously mentioned Weierstraß form is that basic cryptographic primitives require fewer arithmetic operations, resulting in faster cryptographic algorithms (see Bernstein *et al.* in [15] for details).

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points belonging to a twisted Edwards elliptic curve (henceforth known simply as an EC). We define addition on points by defining $P_1 + P_2 = (x_1, y_1) + (x_2, y_2)$ as the point $P_3 = (x_3, y_3)$ where[8]

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \pmod q$$
$$y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2} \pmod q$$

These formulas for addition also apply to point doubling; that is, when $P_1 = P_2$. To subtract a point, invert its coordinates over the y-axis, $(x, y) \to (-x, y)$ [13], and use normal point addition. Recall that 'negative' elements $-x$ of $\mathbb{F}_q$ are really $-x \pmod q$.

Whenever two curve points are added together, P_3 is a point on the 'original' elliptic curve, or in other words all $x_3, y_3 \in \mathbb{F}_q$ and satisfy the EC equation.

[7] Notation: The phrase $a \in \mathbb{F}$ means a is some element in the field $\mathbb{F}$.

[8] Typically elliptic curve points are converted into projective coordinates (or a similar representation, e.g. extended twisted Edwards coordinates [38]) prior to curve operations like point addition, in order to avoid performing field inversions for efficiency. [67]

2.3.2 Group theory

Importantly, elliptic curves have what is known as an *abelian group* structure [1]. Every curve has a so-called 'point-at-infinity' I, which is like a 'zero position' on the curve (its coordinates are $(0,1)$), and a finite number of points N that can be computed. Any point P added with itself N times will produce the point-at-infinity I, and $I + P = P$.

Group theory: Intro

For now, let's step back from elliptic curves and imagine a clock-like ring with *order* N.[9] The zeroth position 0 (or I) is at the top, followed by 1 and preceded by $N-1$. Clearly, we can walk around the ring with step-size 1, and reach the top again after N steps.

While it's useful to think of ourselves as walking around the circle, it's much more accurate to think about each position on the circle as being a 'point', and taking steps is like 'adding points together'. If we stand at point P_2 on a circle with $N = 6$, then with each 'step' we are adding P_2 to the point we are currently standing on.

$$\begin{aligned} I + P_2 &= P_2 \\ P_2 + P_2 &= P_4 \\ P_4 + P_2 &= P_6 = I \end{aligned}$$

Note how we landed back on our starting position after three steps. The point P_2 has generated a *cyclic* subgroup with order 3 out of multiples of itself. It's cyclic because after a while you always get back to where you started. It's a subgroup since it doesn't (necessarily) contain all the points on the circle.

The order of any point is equal to the number of points in the subgroup it can generate. If a point's order is prime, then all the other (non-point-at-infinity) points it generates will generate the same subgroup. In our previous example, where the subgroup's order was 3, the point P_4 also generates the same subgroup.

$$\begin{aligned} I + P_4 &= P_4 \\ P_4 + P_4 &= P_2 \\ P_2 + P_4 &= P_6 = I \end{aligned}$$

However, P_1 has order 6 which is not prime, so not all of the points it generates have order 6 (only P_5 will generate the same subgroup).

Group theory: Useful concepts

We always land somewhere on the ring no matter how many multiples of a point we add together. This lets us simplify scalar multiplication from nP to $[n \pmod{u}]P$, where u is the order of the

[9] The basics of group theory are very important to grasp for the rest of this document. Visual learners may find it helpful to draw pictures and work out what is happening by hand.

point P. A point can't actually be multiplied by 0, so if $n \pmod{u} \stackrel{?}{=} 0$ just multiply by u or return the 0^{th} position directly (I). The orders of all possible subgroups are divisors of N (by *Lagrange's theorem* [44]).

To find the order, u, of any given point P's subgroup:

1. Find N (e.g. use *Schoof's algorithm* [63]).
2. Find all the divisors of N.
3. For every divisor n of N, compute nP.
4. The smallest n such that $nP \stackrel{?}{=} I$ is the order u of the subgroup.

Suppose we are given two points P_a, P_b and are told they both have order u. Do they necessarily belong to the same subgroup, or might there be more than one subgroup with order u?

We can think of the two points in terms of P_1 which has order N, such that $P_a = n_a * P_1$ and $P_b = n_b * P_1$. We know that $N * P_1 = I$, and that $u * P_a = u * P_b = I$. Therefore $u * n_a$ and $u * n_b$ must be multiples of N.

Since u is a divisor of N (recalling Lagrange's theorem), for $u * n_a$ and $u * n_b$ to be multiples of N, scalars n_a, n_b must have a common denominator that is another divisor of N, namely $e = N/u$. Therefore $P_a = (n_a/e) * e * P_1$ and $P_b = (n_b/e) * e * P_1$, or in other words P_a and P_b are multiples of the same point $e * P_1$ and must both be members of that point's subgroup.

Put simply, any two points P_a and P_b with order u are in the same subgroup, which is composed of multiples of $(N/u) * P_1$. Furthermore, for any random point $P' = n' * P_1$, the expression $(N/u) * P'$ will either be a point in the u subgroup (since $n' * (N/u) * P_1$ is a multiple of $(N/u) * P_1$), or I (in which case n' must be a multiple of u, so P' is a member of the $e = (N/u)$ subgroup).

Group theory: Back to elliptic curves

Elliptic curve points have no concept of 'proximity', so for our clock-like example with $N = 6$, P_3 is no 'closer' or 'farther' from I than P_1. However, to connect the analogy we can 'map' curve points onto the ring. Take any point P_w with order N and put it at position 1, then construct the ring out of multiples of P_w. All of the observations we have made so far still hold, and will hold even if the mapping is redone with a different point P_z that also has order N.

ECs selected for cryptography typically have $N = hl$, where l is some sufficiently large (such as 160 bits) prime number and h is the so-called *cofactor* which could be as small as 1 or 2.[10] One point in the subgroup of size l is usually selected to be the generator G as a convention. For every other point P in that subgroup there exists an integer $0 < n \leq l$ satisfying $P = nG$.

[dalek25519] src/backend/serial/curve_models/mod.rs

Based on our understanding from the previous section, we can use the following algorithm to find (non-point-at-infinity) points in the subgroup of order l:

[10] EC with small cofactors allow relatively faster point addition, etc. [13].

1. Find N of the elliptic curve EC, choose subgroup order l, compute $h = N/l$.
2. Choose a random point P' in EC.
3. Compute $P = hP'$.
4. If $P \stackrel{?}{=} I$ return to step 2; otherwise, P is in the subgroup of order l.

Calculating the scalar product between any integer n and any point P, nP, is not difficult, whereas finding n such that $P_1 = nP_2$ is thought to be computationally hard. By analogy to modular arithmetic, this is often called the *discrete logarithm problem* (DLP).[11] Scalar multiplication can be seen as a *one-way function*, which paves the way for using elliptic curves for cryptography.[12]

[dalek25519] src/backend/ serial/ [u32|u64]/ scalar.rs

The scalar product nP is equivalent to $(((P + P) + (P + P))...)$. Though not always the most efficient approach, we can use double-and-add like in Section 2.2.1. To get the sum $R = nP$, remember we use the + point operation discussed in Section 2.3.1.

1. Define $n_{scalar} \rightarrow n_{binary}$; $A = [n_{binary}]$; $R = I$, the point-at-infinity; $S = P$
2. Iterate through: $i = (0, ..., A_{size} - 1)$
   (a) If A[i] $\stackrel{?}{=}$ 1, then R += S.
   (b) Compute S += S.
3. Use the final R as result.

Note that EC scalars for points in the subgroup of size l (which we will be using henceforth) are members of the finite field $\mathbb{F}_l$. This means arithmetic operations between scalars are mod l.

2.3.3 Public key cryptography with elliptic curves

Public key cryptography algorithms can be devised in a way analogous to modular arithmetic.

Let k be a randomly selected number satisfying $0 < k < l$, and call it a *private key*.[13] Calculate the corresponding *public key* K (an EC point) with the scalar product $kG = K$.

Due to the *discrete logarithm problem* (DLP), we cannot easily deduce k from K alone. This property allows us to use the values (k, K) in standard public key cryptography algorithms.

[11] In modular arithmetic, finding the discrete log of h with respect to g, x, such that $g^x = h$, is thought to be difficult for some group orders. [25]

[12] No known equation or algorithm can efficiently (based on available technology) solve for n in $P_1 = nP_2$, meaning it would take many, many years to unravel just one scalar product.

[13] The private key is sometimes known as a *secret key*. This lets us abbreviate: pk = public key, sk = secret key.

2.3.4 Diffie-Hellman key exchange with elliptic curves

A basic *Diffie-Hellman* [24] exchange of a shared secret between *Alice* and *Bob* could take place in the following manner:

1. Alice and Bob generate their own private/public keys (k_A, K_A) and (k_B, K_B). Both publish or exchange their public keys, and keep the private keys for themselves.

2. Clearly, it holds that
$$S = k_A K_B = k_A k_B G = k_B k_A G = k_B K_A$$
Alice could privately calculate $S = k_A K_B$, and Bob $S = k_B K_A$, allowing them to use this single value as a shared secret.

   For example, if Alice has a message m to send Bob, she could hash the shared secret $h = \mathcal{H}(S)$, compute $x = m + h$, and send x to Bob. Bob computes $h' = \mathcal{H}(S)$, calculates $m = x - h'$, and learns m.

An external observer would not be able to easily calculate the shared secret due to the 'Diffie-Hellman Problem' (DHP), which says finding S from K_A and K_B is very difficult. Also, the DLP prevents them from finding k_A or k_B.[14]

2.3.5 Schnorr signatures and the Fiat-Shamir transform

In 1989 Claus-Peter Schnorr published a now-famous interactive authentication protocol [58], generalized by Maurer in 2009 [45], that allows someone to prove they know the private key k of a given public key K without revealing any information about it [47]. It goes something like this:

1. The prover generates a random integer $\alpha \in_R \mathbb{Z}_l$,[15] computes αG, and sends αG to the verifier.

2. The verifier generates a random *challenge* $c \in_R \mathbb{Z}_l$ and sends c to the prover.

3. The prover computes the *response* $r = \alpha + c * k$ and sends r to the verifier.

4. The verifier computes $R = rG$ and $R' = \alpha G + c * K$, and checks $R \stackrel{?}{=} R'$.

The verifier can compute $R' = \alpha G + c * K$ before the prover, so providing c is like saying, "I challenge you to respond with the discrete logarithm of R'." A challenge the prover can only overcome by knowing k (except with negligible probability).

[14] The DHP is thought to be of at least similar difficulty to the DLP, although it has not been proven. [29]

[15] Notation: The R in $\alpha \in_R \mathbb{Z}_l$ means α is randomly selected from $\{1, 2, 3, ..., l-1\}$. In other words, $\mathbb{Z}_l$ is all integers $\pmod{l}$. We exclude 'l' since the point-at-infinity is not useful here.

[dalek25519] src/scalar.rs `Scalar::random()`

If α was chosen randomly by the prover, then r is randomly distributed [59] and k is information-theoretically secure within r (it can still be found by solving the DLP for K or αG).[16] However, if the prover reuses α to prove his knowledge of k, anyone who knows both challenges in $r = \alpha + c * k$ and $r' = \alpha + c' * k$ can compute k (two equations, two unknowns).[17]

$$k = \frac{r - r'}{c - c'}$$

If the prover knew c from the beginning (e.g. if the verifier secretly gave it to her), she could generate a random response r and compute $\alpha G = rG - cK$. When she later sends r to the verifier, she 'proves' knowledge of k without ever having to know it. Someone observing the transcript of events between prover and verifier would be none the wiser. The scheme is not *publicly verifiable.* [47]

In his role as challenger, the verifier spits out a random number after receiving αG, making him equivalent to a *random function.* Random functions, such as hash functions, are known as random oracles because computing one is like requesting a random number from someone [47].[18]

Using a hash function, instead of the verifier, to generate challenges is known as a *Fiat-Shamir transform* [26], because it makes an interactive proof non-interactive and publicly verifiable [47].[19,20]

Non-interactive proof

1. Generate random number $\alpha \in_R \mathbb{Z}_l$, and compute αG.
2. Calculate the challenge using a cryptographically secure hash function, $c = \mathcal{H}(T_p, [\alpha G])$.[21]
3. Define the response $r = \alpha + c * k$.
4. Publish the proof pair $(\alpha G, r)$.

[16] A cryptosystem with information-theoretic security is one where even an adversary with infinite computing power could not break it, because they simply wouldn't have enough information.

[17] If the prover is a computer, you could imagine someone 'cloning'/copying the computer after it generates α, then presenting each copy with a different challenge.

[18] More generally, "[i]n cryptography... an oracle is any system which can give some extra information on a system, which otherwise would not be available." [2]

[19] The output of a cryptographic hash function $\mathcal{H}$ is uniformly distributed across the range of possible outputs. That is to say, for some input A, $\mathcal{H}(A) \in^D_R \mathbb{S}_H$ where $\mathbb{S}_H$ is the set of possible outputs from $\mathcal{H}$. We use $\in^D_R$ to indicate the function is deterministically random. $\mathcal{H}(A)$ produces the same thing every time, but its output is equivalent to a random number.

[20] Note that non-interactive Schnorr-like proofs (and signatures) require either use of a fixed generator G, or inclusion of the generator in the challenge hash. Including it that way is known as key prefixing, which we discuss more later (Sections **??** and **??**).

[21] MobileCoin has a policy of 'domain separating' [65] different uses of hash functions. This in practice means prefixing each 'use case' of a hash function with a unique bit-string. We model it here with the tag T_p, which might be the text string "simple Schnorr proof". Domain separated hash functions have different outputs even with the same inputs. For the remainder of this document we leave out domain separation tags for succinctness, but unless otherwise stated all uses of hash function have their own tag.

[MC-tx] src/domain_separators.rs

Verification

1. Calculate the challenge: $c' = \mathcal{H}(T_p, [\alpha G])$.
2. Compute $R = rG$ and $R' = \alpha G + c' * K$.
3. If $R = R'$ then the prover must know k (except with negligible probability).

Why it works

$$
\begin{aligned}
rG &= (\alpha + c * k)G \\
&= (\alpha G) + (c * kG) \\
&= \alpha G + c * K \\
R &= R'
\end{aligned}
$$

An important part of any proof/signature scheme is the resources required to verify them. This includes space to store proofs and time spent verifying. In this scheme we store one EC point and one integer, and need to know the public key — another EC point. Since hash functions are comparatively fast to compute, keep in mind that verification time is mostly a function of elliptic curve operations.

[dalek25519] src/edwards.rs

2.3.6 Signing messages

Typically, a cryptographic signature is performed on a cryptographic hash of a message rather than the message itself, which facilitates signing messages of varying size. However, in this report we will loosely use the term 'message', and its symbol $\mathfrak{m}$, to refer to the message properly speaking and/or its hash value, unless specified.

Signing messages is a staple of Internet security that lets a message's recipient be confident its content is as intended by the signer. One common signature scheme is called ECDSA. See [39], ANSI X9.62, and [37] for more on this topic.

The signature scheme we present here is an alternative formulation of the transformed Schnorr proof from before. Thinking of signatures in this way prepares us for exploring ring signatures in the next chapter.

Signature

Assume Alice has the private/public key pair (k_A, K_A). To unequivocally sign an arbitrary message $\mathfrak{m}$, she could execute the following steps:

1. Generate random number $\alpha \in_R \mathbb{Z}_l$, and compute αG.

2. Calculate the challenge using a cryptographically secure hash function, $c = \mathcal{H}(\mathfrak{m}, [\alpha G])$.
3. Define the response r such that $\alpha = r + c * k_A$. In other words, $r = \alpha - c * k_A$.
4. Publish the signature (c, r).

Verification

Any third party who knows the EC domain parameters (specifying which elliptic curve was used), the signature (c, r), the signing method, $\mathfrak{m}$, the hash function, and K_A can verify the signature:

1. Calculate the challenge: $c' = \mathcal{H}(\mathfrak{m}, [rG + c * K_A])$.
2. If $c = c'$ then the signature passes.

In this signature scheme we store two scalars, and need to know one public EC key.

Why it works

This stems from the fact that

$$\begin{aligned} rG &= (\alpha - c * k_A)G \\ &= \alpha G - c * K_A \\ \alpha G &= rG + c * K_A \\ \mathcal{H}_n(\mathfrak{m}, [\alpha G]) &= \mathcal{H}_n(\mathfrak{m}, [rG + c * K_A]) \\ c &= c' \end{aligned}$$

Therefore the owner of k_A (Alice) created (c, r) for $\mathfrak{m}$: she signed the message. The probability someone else, a forger without k_A, could have made (c, r) is negligible, so a verifier can be confident the message was not tampered with.

2.4 Curve Ed25519 and Ristretto

MobileCoin uses a particular twisted Edwards elliptic curve for cryptographic operations, *Ed25519*, the *birational equivalent*[22] of the Montgomery curve *Curve25519*. It actually uses Ed25519 indirectly via the Ristretto encoding abstraction, which we will discuss. Both Curve25519 and Ed25519 were released by Bernstein *et al.* [13, 14, 15].

[22] Without giving further details, birational equivalence can be thought of as an isomorphism expressible using rational terms.

The curve is defined over the prime field $\mathbb{F}_{2^{255}-19}$ (i.e. $q = 2^{255} - 19$) by means of the following equation:

[dalek25519] src/const-ants.rs `#[test] test_d_vs_ratio()`

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$$

This curve addresses many concerns raised by the cryptography community.[23] It is well known that NIST[24] standard algorithms have issues. For example, it has recently become clear the NIST standard random number generation algorithm PNRG (the version based on elliptic curves) is flawed and contains a potential backdoor [35]. Seen from a broader perspective, standardization authorities like NIST lead to a cryptographic monoculture, introducing a point of centralization. A great example of this was illustrated when the NSA used its influence over NIST to weaken an international cryptographic standard [5].

Curve Ed25519 is not subject to any patents (see [42] for a discussion on this subject), and the team behind it has developed and adapted basic cryptographic algorithms with efficiency in mind [15].

[dalek25519] src/edw-ards.rs

Twisted Edwards curves have order expressible as $N = 2^c l$, where l is a prime number and c is a positive integer. In the case of curve Ed25519, its order is a 76-digit number (l is 253 bits):[25]

$$2^3 \cdot 7237005577332262213973186563042994240857116359379907606001950938285454250989$$

[dalek25519] src/const-ants.rs `BASEPOINT_ORDER`

2.4.1 Problem with cofactors

As mentioned in Section 2.3.2, only points in the large prime-order subgroup of a given elliptic curve are used in cryptographic algorithms. It is therefore sometimes important to make sure a given curve point belongs to that subgroup [36].

For example, it is possible to add a point from the subgroup of size h (the cofactor) to a point P, and, with a scalar n that is a multiple of h, create several points which when multiplied by n have the same resultant point. This is because an EC point multiplied by its subgroup's order is 'zero'.[26]

To be clear, given some point K in the subgroup of order l, any point K^h with order h, and an integer n divisible by h:

$$\begin{aligned} n * (K + K^h) &= nK + nK^h \\ &= nK + 0 \end{aligned}$$

[23] Even if a curve appears to have no cryptographic security problems, it's possible the person/organization that created it knows a secret issue which only crops up in very rare curves. Such a person may have to randomly generate many curves in order to find one with a hidden weakness and no known weaknesses. If reasonable explanations are required for curve parameters, then it becomes even more difficult to find weak curves that will be accepted by the cryptographic community. Curve Ed25519 is known as a 'fully rigid' curve, which means its generation process was fully explained. [57]

[24] National Institute of Standards and Technology, https://www.nist.gov/.

[25] This means private EC keys in Ed25519 are 253 bits.

[26] Cryptocurrencies that inherited the CryptoNote code base had an infamous vulnerability related to adding cofactor-subgroup points to normal-subgroup points. It was solved by checking that key images (discussed in Chapter ??) are in the correct subgroup with the test $l\tilde{K} \stackrel{?}{=} 0$ [28].

The importance of using prime-order-subgroup points was the motivation behind Ristretto, which is an encoding abstraction for twisted Edwards-based curves that efficiently constructs a prime-order group out of the underlying non-prime group. [34]

2.4.2 Ristretto

Ristretto can be thought of as a 'binning' procedure for twisted Edwards curve points. The number of bins is equal to the prime-order subgroup of the relevant curve (l), and each bin has h elements.[27] Curve operations 'on' or 'between' bins behave just like curve operations on the prime-order subgroup, except in this model rather than a specific point the result is a specific bin.

This implies the members of a bin must be variants of a prime-order subgroup point (in other words, a prime-order point plus all the members of the cofactor-order subgroup, including the point-at-infinity). Given two bins, adding any of their members together will land you in the same third bin. In this way curve operations on 'Ristretto points', which are simple containers that can hold a bin member from any bin, behave just like operations on prime-order subgroup points.

For example, given Ristretto points

$$P_1 = P_1^{prime} + P_1^{cofactor}$$
$$P_2 = P_2^{prime} + P_2^{cofactor}$$

their sum $P_1 + P_2 = P_3$ will be

$$P_3^{prime} = P_1^{prime} + P_2^{prime}$$
$$P_3^{cofactor} = P_1^{cofactor} + P_2^{cofactor}$$

Here P_3^{prime} defines which bin you landed in, and $P_3^{cofactor}$ corresponds to the specific member that was created.

Two members of the same bin are considered 'equal'. There is a relatively cheap way to test equality, which we describe in Section 2.4.4.

The important innovation of Ristretto is each bin has a representative 'canonical member' that can be easily found by 'compressing' any of the bin members and then 'decompressing' the result. This way curve points can be communicated in compressed form, and recipients can be assured they are handling effectively prime-order points and don't have to be concerned about cofactor-related problems.

As a bit of callback, given some point K in the prime-order subgroup and two points K_a^h, K_b^h in the cofactor-order subgroup, both $K + K_a^h$ and $K + K_b^h$ will compress and then decompress into the same point $K + K_c^h$ (where c may equal a or b, or be a different cofactor point).

Since all bin members get compressed to the same bit string, there is no 'gotcha' (as there is with standard Ed25519) where presenting different compressed members of a bin to a byte-aware context (e.g. a hash function or byte-wise comparison) will have different results.

[27] In group theory, what we call bins are more correctly known as 'cosets' [68].

2.4.3 Binary representation

Elements of $\mathbb{F}_{2^{255}-19}$ are encoded as 256-bit integers, so they can be represented using 32 bytes. Since each element only requires 255 bits, the most significant bit is always zero.

Consequently, any point in Ed25519 could be expressed using 64 bytes. By applying the Ristretto *point compression* technique, described below, however, it is possible to reduce this amount by half, to 32 bytes.

2.4.4 Point compression

The Ed25519 curve has the property that its points can be easily compressed, so that representing a point will consume only the space of one coordinate. We will not delve into the mathematics necessary to justify this [31], but we can give a brief insight into how it works. Normal point compression for the Ed25519 curve was standardized in [40], first described in [14], and the concept was introduced in [50].

As background, it's helpful to know the normal point compression scheme follows from a transformation of the twisted Edwards curve equation (wherein $a = -1$): $x^2 = (y^2 - 1)/(dy^2 + 1)$,[28] which indicates there are two possible x values ($+$ or $-$) for each y. Field elements x and y are calculated $\pmod q$, so there are no actual negative values. However, taking $\pmod q$ of $-x$ will change the value between odd and even since q is odd. For example: $3 \pmod 5 = 3$, $-3 \pmod 5 = 2$. In other words, the field elements x and $-x$ have different odd/even assignments.

If we have a curve point and know its x is even, but given its y value the transformed curve equation outputs an odd number, then we know negating that number will give us the right x. One bit can convey this information, and conveniently the y coordinate has an extra bit.

Ristretto has a different approach to compressing points, where the sign of the x coordinate is not encoded.

Assume we want to compress a point (x, y). First we transform it into *extended twisted Edwards coordinates* [38] $(X : Y : Z : T)$, where $XY = ZT$ and $aX^2 + Y^2 = Z^2 + dT^2$.

$$\begin{aligned} X &= x \\ Y &= y \\ Z &= 1 \\ T &= xy \pmod q \end{aligned}$$

Square Root: `Sqrt(u, v)`

1. Create an algorithm for computing $\sqrt{u/v} \pmod q$.[29]

[dalek25519] src/field.rs `FieldElement::sqrt_ratio_i()`

[28] Here $d = -\frac{121665}{121666}$.

[29] These algorithms are merely shown for a sense of completeness. It's best to consult the `dalek` library's implementation and notes [31] for any production-level applications.

2. Compute[30] $z = uv^3(uv^7)^{(q-5)/8} \pmod q$.
   (a) If $vz^2 \stackrel{?}{=} u \pmod q$, set $r = z$.
   (b) If $vz^2 \stackrel{?}{=} -u \pmod q$, calculate $r = z * 2^{(q-1)/4} \pmod q$.
3. If the least significant bit of r is 1 (i.e. it is odd), return $-r$, otherwise return r.[31]

Encoding

[dalek25519] src/ristretto.rs `RistrettoPoint::compress()`

1. Define
   (a) $u_1 = (Z + Y) * (Z - Y) \pmod q$
   (b) $u_2 = XY \pmod q$
2. Let inv $= \mathtt{Sqrt}(1, u_1 u_2^2) \pmod q$.
3. Define
   (a) $i_1 = u_1 * \text{inv} \pmod q$
   (b) $i_2 = u_2 * \text{inv} \pmod q$
   (c) $z_{inv} = i_1 i_2 T \pmod q$
4. Let b equal the least significant bit of $z_{inv} * T \pmod q$
   (a) If $b \stackrel{?}{=} 1$, define
      i. $X' = Y * 2^{(q-1)/4} \pmod q$
      ii. $Y' = X * 2^{(q-1)/4} \pmod q$
      iii. $D' = i_1 * \mathtt{Sqrt}(1, a - d) \pmod q$
   (b) Otherwise if $b \stackrel{?}{=} 0$, define
      i. $X' = X$
      ii. $Y' = Y$
      iii. $D' = i_2$
5. If $z_{inv} * X' \pmod q$ is odd, set $Y' = -Y' \pmod q$.
6. Compute $s = D' * (Z - Y') \pmod q$. If s is odd, set $s = -s \pmod q$.
7. Return s.

Decoding

[dalek25519] src/ristretto.rs `CompressedRistretto::decompress()`

1. Given a supposed compressed curve point s, check if it is a valid field element with a byte-wise comparison $s \pmod q \stackrel{?}{=} s$. Reject s if it is odd or invalid.

[30] Since $q = 2^{255} - 19 \equiv 5 \pmod 8$, $(q-5)/8$ and $(q-1)/4$ are straightforward integers.

[31] According to the comments in `[dalek] src/field.rs FieldElement::sqrt_ratio_i()`, only the 'positive' square root should be returned, which is defined by convention in [14] as field elements with the least significant bit not set (i.e. 'even' field elements). In normal Ed25519 point decompression [14] we would compute $\mathtt{Sqrt}(y^2 - 1, dy^2 + 1)$, then decide whether to use the 'positive'/'negative' result variant depending on if we want the even/odd x coordinate. Basically, a compressed point is the y coordinate, with the most significant bit equal to 0 or 1 to indicate if the point's x coordinate is even/odd.

2. Compute
$$y = \frac{1+as^2}{1-as^2} \pmod q$$
3. Compute (x should be 'even' after this step)
$$x = \texttt{Sqrt}(4s^2, ad(1+as^2)^2 - (1-as^2)^2) \pmod q$$
4. Convert to extended coordinates if desired.
$$\begin{aligned} X &= x \\ Y &= y \\ Z &= 1 \\ T &= xy \pmod q \end{aligned}$$

We can use extended coordinates to test if two points belong to the same Ristretto bin. If either $X_1Y_2 \stackrel{?}{=} Y_1X_2$ or $Y_1Y_2 \stackrel{?}{=} -aX_1X_2$ holds, then the points $P_1 = (X_1 : Y_1 : Z_1 : T_1)$ and $P_2 = (X_2 : Y_2 : Z_2 : T_2)$ are 'equal' for our purposes [32].[32,33]

[dalek25519] src/ristretto.rs `RistrettoPoint::ct_eq()`

Implementations of Ed25519 typically use the generator $G = (x, 4/5)$ [14], where x is the 'even' variant based on normal point decompression (footnote 31 from earlier in this section describes how it works) of $y = 4/5 \pmod q$. The Ristretto generator is straightforwardly the bin that contains G, and the point selected to represent it is G itself.

[dalek25519] src/constants.rs `RISTRETTO_BASEPOINT_POINT`

2.5 Binary operator XOR

The binary operator XOR is a useful tool that will appear in Section ??. It takes two arguments and returns true if one, but not both, of them is true [6]. Here is its truth table:

A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F

In the context of computer science, XOR is equivalent to bit addition modulo 2. For example, the XOR of two bit pairs:
$$\begin{aligned} \text{XOR}(\{1,1\},\{1,0\}) &= \{1+1, 1+0\} \pmod 2 \\ &= \{0,1\} \end{aligned}$$

[32] Since $a = -1$, the second test simplifies to $Y_1Y_2 \stackrel{?}{=} X_1X_2$.
[33] Multiple extended coordinates can represent a given curve point [33], so X may not always equal x, and the same for Y and y. This equality test works for all extended coordinate representations.

Each of these also produce $\{0,1\}$: $\text{XOR}(\{1,0\},\{1,1\})$, $\text{XOR}(\{0,0\},\{0,1\})$, and $\text{XOR}(\{0,1\},\{0,0\})$. For XOR inputs with b bits, there are 2^{b} total combinations of inputs that would make the same output. This means if $C = \text{XOR}(A, B)$ and input $A \in_R \{0, ..., 2^{\text{b}} - 1\}$, an observer who learns C would gain no information about B (its real value could be any of 2^b possibilities).

At the same time, anyone who knows two of the elements in $\{A, B, C\}$, where $C = \text{XOR}(A, B)$, can calculate the third element, such as $A = \text{XOR}(B, C)$. XOR indicates if two elements are different or the same, so knowing C and B is enough to expose A. A careful examination of the truth table reveals this vital feature.[34]

[34] One interesting application of XOR (unrelated to MobileCoin) is swapping two bit registers without a third register. We use the symbol $\oplus$ to indicate an XOR operation. $A \oplus A = 0$, so after three XOR operations between the registers: $\{A, B\} \rightarrow \{[A \oplus B], B\} \rightarrow \{[A \oplus B], B \oplus [A \oplus B]\} = \{[A \oplus B], A \oplus 0\} = \{[A \oplus B], A\} \rightarrow \{[A \oplus B] \oplus A, A\} = \{B, A\}$.

Bibliography

[1] Abelian Group. Brilliant.org. https://brilliant.org/wiki/abelian-group/ [Online; accessed 10/03/2020].

[2] cryptography - What is a cryptographic oracle? https://security.stackexchange.com/questions/10617/what-is-a-cryptographic-oracle [Online; accessed 04/22/2018].

[3] Extended Euclidean Algorithm. Brilliant.org. https://brilliant.org/wiki/extended-euclidean-algorithm/ [Online; accessed 12/08/2020].

[4] Fermat's Little Theorem. Brilliant.org. https://brilliant.org/wiki/fermats-little-theorem/ [Online; accessed 12/02/2020].

[5] Trust the math? An Update. http://www.math.columbia.edu/~woit/wordpress/?p=6522 [Online; accessed 04/04/2018].

[6] XOR – from Wolfram Mathworld. http://mathworld.wolfram.com/XOR.html [Online; accessed 04/21/2018].

[7] Fungible, July 2014. https://wiki.mises.org/wiki/Fungible [Online; accessed 03/31/2020].

[8] NIST Releases SHA-3 Cryptographic Hash Standard, August 2015. https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard [Online; accessed 06/02/2018].

[9] Kurt M. Alonso and Jordi Herrera Joancomartí. Monero — Privacy in the Blockchain. Cryptology ePrint Archive, Report 2018/535, 2018. https://eprint.iacr.org/2018/535.

[10] Kurt M. Alonso and koe. Zero to Monero — First Edition, June 2018. https://web.getmonero.org/library/Zero-to-Monero-1-0-0.pdf [Online; accessed 01/15/2020].

[11] Jean-Philippe Aumasson, Willi Meier, Raphael Phan, and Luca Henzen. *The Hash Function BLAKE*. Springer Publishing Company, Incorporated, 2014.

[12] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5, January 2013. https://blake2.net/blake2.pdf [Online; accessed 10/06/2020].

[13] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. *Twisted Edwards Curves*, pages 389–405. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. https://eprint.iacr.org/2008/013.pdf [Online; accessed 02/13/2020].

[14] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, Sep 2012. https://ed25519.cr.yp.to/ed25519-20110705.pdf [Online; accessed 03/04/2020].

[15] Daniel J. Bernstein and Tanja Lange. *Faster Addition and Doubling on Elliptic Curves*, pages 29–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. https://eprint.iacr.org/2007/286.pdf [Online; accessed 03/04/2020].

[16] Karina Bjørnholdt. Dansk politi har knækket bitcoin-koden, May 2017. http://www.dansk-politi.dk/artikler/2017/maj/dansk-politi-har-knaekket-bitcoin-koden [Online; accessed 04/04/2018].

[17] Dan Boneh and Victor Shoup. A Graduate Course in Applied Cryptography. https://crypto.stanford.edu/~dabo/cryptobook/ [Online; accessed 12/30/2019].

[18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. https://eprint.iacr.org/2017/1066.pdf [Online; accessed 10/28/2018].

[19] Chainalysis. THE 2020 STATE OF CRYPTO CRIME, January 2020. https://go.chainalysis.com/rs/503-FAP-074/images/2020-Crypto-Crime-Report.pdf [Online; accessed 02/11/2020].

[20] Victor Costan and Srinivas Devadas. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086 [Online; accessed 10/27/2020].

[21] dalek cryptography. bulletproofs [library]. https://github.com/dalek-cryptography/curve25519-dalek [Online; accessed 10/06/2020].

[22] dalek cryptography. bulletproofs [library]. https://github.com/dalek-cryptography/bulletproofs [Online; accessed 10/06/2020].

[23] RustCrypto Developers. RustCrypto: BLAKE2 [library]. https://docs.rs/crate/blake2/0.9.0 [Online; accessed 10/06/2020].

[24] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006. https://ee.stanford.edu/~hellman/publications/24.pdf [Online; accessed 03/04/2020].

[25] Changyu Dong. Math in Network Security: A Crash Course; Discrete Logarithm Problem. https://www.doc.ic.ac.uk/~mrh/330tutor/ch06s02.html [Online; accessed 11/26/2020].

[26] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. https://link.springer.com/content/pdf/10.1007%2F3-540-47721-7_12.pdf [Online; accessed 03/04/2020].

[27] Ryo "fireice_uk" Cryptocurrency. On-chain tracking of Monero and other Cryptonotes, April 2019. https://medium.com/@crypto_ryo/on-chain-tracking-of-monero-and-other-cryptonotes-e0afc6752527 [Online; accessed 03/25/2020].

[28] Riccardo "fluffypony" Spagni and luigi1111. Disclosure of a Major Bug in Cryptonote Based Currencies, May 2017. https://getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html [Online; accessed 04/10/2018].

[29] Steven Galbraith. Mathematics of Public Key Cryptography [v2.0 Chapter 21 extended version], 2012. https://www.math.auckland.ac.nz/~sgal018/crypto-book/ch21.pdf [Online; accessed 12/07/2020].

[30] Joshua Goldbard and Moxie Marlinspike. MobileCoin, November 2017. https://static.coinpaprika.com/storage/cdn/whitepapers/4235403.pdf?__cf_chl_jschl_tk__=4f3d03da0f296c2787897377ab0da28fdf6abece-1605734577-0-ARyN9UM60LpEfrsQ3ltzLLl__jZTGHp4slF4DQHrW68JD3cCb-4oYKgWr6bHduNKB5A_qhENHM0LZAXG4k9wWAB0XNIBdmwsdaW8CeOmTRQhE06BsL6FII7hTvaN4SrdrNjFmuTYGp43IqKtkg4LK5duDR59GpglMgTD5UObi3Frxp8AmcbGeCyuw_Ki16QOOlMkFM-oY4pydw3DfaHWNjmN4ts5SHXW38st0XVHCQi_M__wzr9PtEG3BnBzQsmFDmESrQRZIYQXc_NpDNzpTPihGzETUVL6okA0zTFuIjX6ztkhxnTT4au6BSTzEA2KZpR7s_ts1qlZ6aw1qp7vsqqxBUY9W-si68n0hNd7ro_wGQuVe7BaNXGwhCOIDfRvYRqKP2vitUCG-_Gs4hkpwC1bt6w [Online; accessed 12/02/2020].

[31] The Ristretto Group. Ristretto. https://ristretto.group/ristretto.html [Online; accessed 10/05/2020].

[32] The Ristretto Group. Ristretto in Detail: Equality Testing. https://ristretto.group/details/equality.html [Online; accessed 10/06/2020].

[33] The Ristretto Group. What is Ristretto? https://ristretto.group/what_is_ristretto.html [Online; accessed 10/06/2020].

[34] The Ristretto Group. Why Ristretto? https://ristretto.group/why_ristretto.html [Online; accessed 10/03/2020].

[35] Thomas C. Hales. The NSA back door to NIST. *Notices of the AMS*, 61(2):190–192. https://www.ams.org/notices/201402/rnoti-p190.pdf [Online; accessed 03/04/2020].

[36] Mike Hamburg. Decaf: Eliminating cofactors through point compression, 2015. https://www.shiftleft.org/papers/decaf/decaf.pdf [Online; accessed 10/05/2020].

[37] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[38] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards Curves Revisited. Cryptology ePrint Archive, Report 2008/522, 2008. https://eprint.iacr.org/2008/522 [Online; accessed 10/05/2020].

[39] Don Johnson and Alfred Menezes. The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical Report CORR 99-34, Dept. of C&O, University of Waterloo, Canada, 1999. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.9475&rep=rep1&type=pdf [Online; accessed 04/04/2018].

[40] S. Josefsson, SJD AB, and N. Moeller. EdDSA and Ed25519. Internet Research Task Force (IRTF), 2015. https://tools.ietf.org/html/draft-josefsson-eddsa-ed25519-03 [Online; accessed 05/11/2018].

[41] Bradley Kjell. Big Endian and Little Endian. https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html [Online; accessed 01/23/2020].

[42] Alexander Klimov. ECC Patents?, October 2005. http://article.gmane.org/gmane.comp.encryption.general/7522 [Online; accessed 04/04/2018].

[43] koe, Kurt M. Alonso, and Sarang Noether. Zero to Monero — Second Edition, April 2020. https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf [Online; accessed 10/03/2020].

[44] Ben Lynn. Lagrange's Theorem. https://crypto.stanford.edu/pbc/notes/group/lagrange.html [Online; accessed 12/02/2020].

[45] Ueli Maurer. Unifying Zero-Knowledge Proofs of Knowledge. In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, pages 272–286, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. https://www.crypto.ethz.ch/publications/files/Maurer09.pdf [Online; accessed 03/04/2020].

[46] Gregory Maxwell. Confidential Transactions. https://elementsproject.org/features/confidential-transactions/investigation [Online; accessed 11/23/2020].

[47] Gregory Maxwell and Andrew Poelstra. Borromean Ring Signatures. 2015. https://pdfs.semanticscholar.org/4160/470c7f6cf05ffc81a98e8fd67fb0c84836ea.pdf [Online; accessed 04/04/2018].

[48] David Mazières. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, February 2016. https://www.stellar.org/papers/stellar-consensus-protocol?locale=en [Online; accessed 12/02/2020].

[49] R. C. Merkle. Protocols for Public Key Cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122, April 1980. http://www.merkle.com/papers/Protocols.pdf [Online; accessed 03/04/2020].

[50] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, Berlin, Heidelberg, 1986. Springer-Verlag. https://link.springer.com/content/pdf/10.1007/3-540-39799-X_31.pdf [Online; accessed 03/04/2020].

[51] Nicola Minichiello. The Bitcoin Big Bang: Tracking Tainted Bitcoins, June 2015. https://bravenewcoin.com/insights/the-bitcoin-big-bang-tracking-tainted-bitcoins [Online; accessed 03/31/2020].

[52] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. http://bitcoin.org/bitcoin.pdf [Online; accessed 03/04/2020].

[53] Arvind Narayanan and Malte Möser. Obfuscation in Bitcoin: Techniques and Politics. *CoRR*, abs/1706.05432, 2017. https://arxiv.org/ftp/arxiv/papers/1706/1706.05432.pdf [Online; accessed 03/04/2020].

[54] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring Confidential Transactions, MRL-0005, February 2016. https://web.getmonero.org/resources/research-lab/pubs/MRL-0005.pdf [Online; accessed 06/15/2018].

[55] Michael Padilla. Beating Bitcoin bad guys, August 2016. http://www.sandia.gov/news/publications/labnews/articles/2016/19-08/bitcoin.html [Online; accessed 04/04/2018].

[56] Jamie Redman. Industry Execs Claim Freshly Minted 'Virgin Bitcoins' Fetch 20% Premium, March 2020. https://news.bitcoin.com/industry-execs-freshly-minted-virgin-bitcoins/ [Online; accessed 03/31/2020].

[57] SafeCurves. SafeCurves: choosing safe curves for elliptic-curve cryptography, 2013. https://safecurves.cr.yp.to/rigid.html [Online; accessed 03/25/2020].

[58] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York. https://link.springer.com/content/pdf/10.1007%2F0-387-34805-0_22.pdf [Online; accessed 03/04/2020].

[59] Paola Scozzafava. Uniform distribution and sum modulo m of independent random variables. *Statistics & Probability Letters*, 18(4):313 – 314, 1993. https://sci-hub.tw/https://www.sciencedirect.com/science/article/abs/pii/016771529390021A [Online; accessed 03/04/2020].

[60] Bassam El Khoury Seguias. Monero Building Blocks, 2018. https://delfr.com/category/monero/ [Online; accessed 10/28/2018].

[61] QingChun ShenTu and Jianping Yu. Research on Anonymization and De-anonymization in the Bitcoin System. *CoRR*, abs/1510.07782, 2015. https://arxiv.org/ftp/arxiv/papers/1510/1510.07782.pdf [Online; accessed 03/04/2020].

[62] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*, pages 15–16. Cambridge University Press, 2005.

[63] Andrew V. Sutherland. Schoof's algorithm, March 2015. https://math.mit.edu/classes/18.783/2015/LectureNotes9.pdf [Online; accessed 12/02/2020].

[64] thankful_for_today. [ANN][BMR] Bitmonero - a new coin based on CryptoNote technology - LAUNCHED, April 2014. Monero's actual launch date was April 18^{th}, 2014. https://bitcointalk.org/index.php?topic=563821.0 [Online; accessed 05/24/2018].

[65] Martin Thomson, Daniel Gillmore, and Benjamin Kaduk. Using Context Labels for Domain Separation of Cryptographic Objects, May 2016. http://www.watersprings.org/pub/id/draft-thomson-saag-context-labels-00.html [Online; accessed 10/08/2020].

[66] Nicolas van Saberhagen. CryptoNote V2.0. https://cryptonote.org/whitepaper.pdf [Online; accessed 04/04/2018].

[67] Wikibooks. Cryptography/Prime Curve/Standard Projective Coordinates, March 2011. https://en.wikibooks.org/wiki/Cryptography/Prime_Curve/Standard_Projective_Coordinates [Online; accessed 03/03/2020].

[68] Wikibooks. Undergraduate Mathematics/Coset, 2020. https://en.wikibooks.org/wiki/Undergraduate_Mathematics/Coset [Online; accessed 10/06/2020].