Corretto: Short ring signatures with Bulletproofs

Carlos Perez
Dusk Foundation*
carlos@dusk.network

Luke Pearson
Dusk Foundation
luke@dusk.network

April 2019

Abstract

Corretto is one of the cryptographic protocols built by, and implemented within the Dusk Network, which uses zero-knowledge proofs to show the existence of a private key within one of many public keys. For this, there will be an elliptic curve, used for the key generation, defined over a Ristretto scalar field which enables the use of Ristretto in Bulletproofs while simultaneously abstracting the computationally intensive conversion within Rank-1 Constraint System from co-factor 8 scalar field into a co-factor 1 Ristretto field. This paper provides an explanation of the current curve development, as well as a contextual understanding of how this curve implementation acts as one of aspects within the Corretto protocol.

---

*https://dusk.network/

Contents

1 Introduction **3**

2 Set Inclusion **3**
2.1 Example 3
2.2 Advantages 4

3 Bulletproofs and Rank 1 Constraint Circuits **4**

4 The Ristretto Scalar Field **5**

5 Equations **7**
5.1 Twisted Edwards and Montgomery Forms 7
5.2 Weierstrass Form 9

6 Field Elements **9**

1 Introduction

The construction and use of elliptic curves is paramount to many cryptographic protocols. Elliptic curves are among the fastest performing primitives where the discrete logarithm problem is hard, which is why they are regarded dominant in the field of cryptography. As the field of cryptography advances, elliptic curves have been proved to be unparalleled in their use as a cryptographic system at which speed and security are two of the most outstanding features. Corretto uses elliptic curves for both private and public key generation, these keys are used in conjunction with zero-knowledge proofs to show how they relate to one another. By creating proofs which show a private key exists in one of many public keys, it is possible to perform cryptographic functions on just the private key, which requires significantly less computational effort but is applicable to all of the keys. To understand the theory and some of the practical applications of elliptic curves and their operations in a stand alone context, or their workings with other cryptographic tools, it is important to familiarise with the difference in utility of various elliptic curves. From a greater understanding of these curves, many of the goals in this current project of Dusk will become apparent. The curve implementation and the choice for certain novel methods can be seen holistically with all the aspects discussed in this paper and as part of a wider pragmatic solution to one of the aspects of the Dusk network, which is to perform elliptic curve cryptography inside of a circuit.
All of the work associated, both current and future either is or will be written in Rust, as this is the language of the library that is being built.

2 Set Inclusion

Set inclusion will be used to show that a private key is one of many public keys. The private and public keys are two generated values from a one way function which are used in a cryptographic system. In any such system, the public key is used for encryption whereas only the private key is used for decryption. A public key is classed as set element, where the set is all of the curve points generated by a base point. The use of set inclusion is to prove that the private key exists as one of many public keys. A set can qualify as a subset of the other set if and only if the elements of the former set are likewise present, yet not the sole elements of the latter set. In order to produce a set inclusion proof, the Prover $\mathcal{P}$ has to convince the Verifier $\mathcal{V}$ that a given set is a subset of another set.

2.1 Example

A simplistic example of the logic outlined above is demonstrated hereafter. If:

$$A = 1, 3, 5$$

$$B = 1, 5$$

then $\boldsymbol{B}$ is a subset, or ***'proper subset' of** $\boldsymbol{A}$**.** It is also important to note that if:

$$B = 1, 3, 5$$

then $\boldsymbol{B}$ **would not be a subset of** $\boldsymbol{A}$ **as** $\boldsymbol{B} = \boldsymbol{A}$, in this case. Also, if

$$B = 1, 4$$

then $\boldsymbol{B}$ **would be a subset but not a proper subset of** $\boldsymbol{A}$, as every element of a B must simultaneously be part of A for the subset to exist.

2.2 Advantages

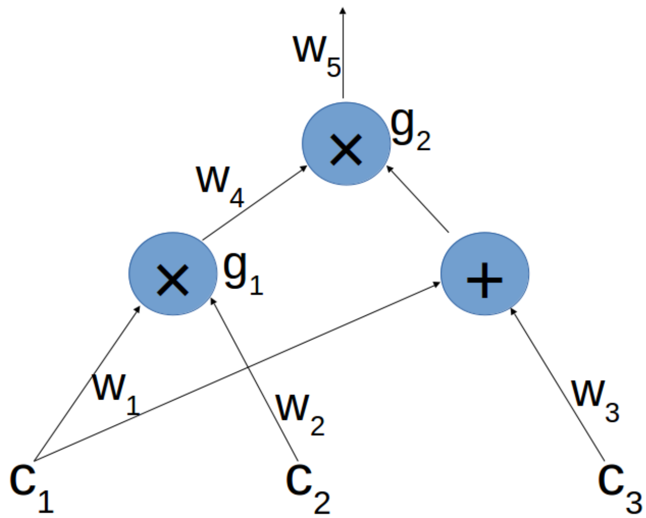
- The advantage of using subsets is that they have varying mathematical properties, the one which is most pertinent to us is the proof that a subset exists inside of a set.
- From this, operations can be performed to that particular subset which can be used to show properties and create proofs of the larger subset without the extra expense as the whole set is not being used.

A full comprehension of this subset rule is very helpful, as well as largely applicable to the defined curve.
For the current set inclusion use case, due to the set elements being public keys and the input being a private key, there needs to be a $ScalarBaseMult(P = x \cdot G)$ operation.

3 Bulletproofs and Rank 1 Constraint Circuits

Bulletproofs[1] are short non-interactive zero-knowledge proofs[2]. For example, Bulletproofs can be used to prove that an encrypted number is in a given range, without leaking any information about the number. Compared to SNARKs[2], Bulletproofs require no trusted setup, which further reduces the risk of a malicious set up. However, Bulletproofs verification is computationally more intensive relative to the SNARK proof verification. Bulletproofs, in context to their computational intensity, have linear scaling, which is measured as the size of the arithmetic circuit.
Bulletproofs are designed to enable efficient confidential transactions in Bitcoin and other cryptocurrencies. Every transaction contains a cryptographic proof which proves the validity of the spending transaction. Bulletproofs shrink the size of the cryptographic proofs from over 10kB to less than 1kB. To prevent overflows every confidential transaction must carry a proof that all amounts are positive and smaller than a threshold. Such range proofs are much smaller with Bulletproofs, this also allows for m transactions to have valid range proofs.
Bulletproofs have many other applications in cryptographic protocols, such as shortening proofs of solvency, short verifiable shuffles, confidential smart contracts, and as a general drop-in replacement for Sigma-protocols.

Bulletproofs are an optimization to the *Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting* paper. The aforementioned paper introduced an inner-product argument by the following diagram.



The constraint system has the following format:

- An vector of n multiplications that gives $3 \cdot n$ low-level variables: left, right and output
- An vector of q linear constraints between these variables.
- Additional m *high-level variables* that represent external facts.

Although Bulletproof implementation provides a solid means of creating fast proofs, the prior choice of curve is important to ensure that binary decomposition is not needed within the circuit for reduction. This reduction is negated as the curve is defined over the Ristretto scalar field.

4 The Ristretto Scalar Field

Ristretto [4] is a technique that constructs prime-order elliptic curve groups, the construction of these groups stems from non prime-order Elliptic curves. Ristretto builds upon the Decaf paper[5], where prime-order curve groups are created from curves with co-factor 4. Ristretto, on the other hand, is applicable to Edwards curve groups which have a co-factor 4 or 8. Edwards curve have a point of order 4, this means that points on the curve are not of prime order and they instead have a small co-factor. By using the Ristretto technique the abstraction problem is solved for all potential co-factor related issues with a single protocol. For use of the Ristretto scalar field in this implementation, any

chosen curve needs to be defined over the Ristretto scalar field, for the prime-order group Ristretto255. This Ristretto scalar field provides a prime-order group of size 2^{252} [4] by encoding group elements. The ristretto255 group will be implemented using points from the curve defined in the next section. This protocol compresses the co-factor of a curve, with the rationale of being able to avoid the drawbacks that are concurrent with a co-factor, whilst being able to capitalize on the robustness of an otherwise solid curve.
If a curve given in standard elliptic curve form, defined as:

- $Y = X^3 + Ax + B$

  then

- Let G be a group of prime order for the curve, denoted as q
- A co-factor, denoted by h, exists such that the order of the curve is $h \cdot q$ for the large prime q

There are various advantages and disadvantages to having a co-factor larger than one, therefore a thorough analysis must be performed, so that it is known whether or not co-factor manipulation is needed. For all curves, except for Hessian curves, the co-factor is divisible by 4. To become more useful to a broad spectrum of cryptography, Ristretto is apt for a large number of curves, which have a co-factor of 8 or 4. When the co-factor is greater than 1 multiple operations can be hindered. In the case of set inclusion, having a co-factor larger than one will hinder the curve operations, specifically relating to the scalar base operations. In reference to the need for subset proofs, the goal is obstructed where the co-factor is not compressed, which leads to non-injective behaviour between the groups. Non-injective functions in set mappings, which is a method to describe whether an element exists in another set or not, affects the operations in proving subsets exists within sets.

For elliptic curves, any scalar multiplication is a 1 to 1 mapping if the group order is prime. Only in a prime-order group is a random scalar for the operation valid, and it must be in the range 1 to q-1. Whereas in a non prime-order group, the adding of a small element can lead to a small subgroup confinement attack[6], which makes it possible to present the same result from different inputs. When implemented, Ristretto acts as a thin layer, which provides a protocol to construct a prime-order group.

To embed a curve into this prime field, the definition that an embedded curve L, is a curve whose base field is defined by the scalar field of another curve, M. In this case, the Doppio curve, which will be eluded to shortly, has a base field which is equal to the scalar field defined by Ristretto255. To visualise how this protocol is performed, when the curve is embedded into the Ristretto scalar field - two arbitrary Edwards points, P and Q, may be represented as the equivalent Ristretto points in the Ristretto scalar field. This happens because the Edwards curve is defined over said field. As a method of creating equivalent points, is not dissimilar to how X, Y, and Z projective coordinates can represent the same P and Q Edwards points for a given Edwards curve. The elements of the created prime-order group, ristretto255, are not curve points, they are simply represented by curve points. For computation understanding, it must be noted that not this prime-order group is not a subgroup of the curve and that there is an unequivocal distinction between the curve points and group elements.

5 Equations

5.1 Twisted Edwards and Montgomery Forms

In order for a selected elliptic align with the goals defined in this paper, it needs to be both twist secure and Ristretto-ready. The Doppio curve has been chosen for the reasons highlighted above.

Which is defined as follows:

- Curve equation
$$-x^2 + y^2 = 1 - \frac{86649}{86650}x^2y^2$$
Which is Twisted Edwards and used to implement Ristretto255.
- $a = -1$
- $d = \frac{86649}{86650}$
- $Basepoint : Y = \frac{8}{9}$
- Montgomery form equivalent:
$$y^2 = x^3 + 346598x^2 + x$$
- $A = 346598$
- $Basepoint : X = 17$
- The number of points on the curve, G, is
$$2^{252} - 121160309657751286123858757838224683208$$
- The prime order of the subgroup, q, is
$$2^{249} - 15145038707218910765482344729778085401$$
- The prime order of the Ristretto scalar field, l, is
$$2^{252} + 27742317777372353535851937790883648493$$
- $Cofactor : h = \frac{G}{q} = 8$

.

5.2 Weierstrass Form

- Weierstrass form equivalent:

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

- $a =$ 2412335192444087404657728854347664746952372119793302535333983646055108025796
- $b =$ 1340186218024493002587627141304258192751317844329612519629993998710484804961

The computation of the Weierstrass form is made to prove point addition in the simplest possible form as this underlines all of the current elliptic curve operations. These initial operations on the field elements are inline, which is made to ensure the most efficient computation possible.

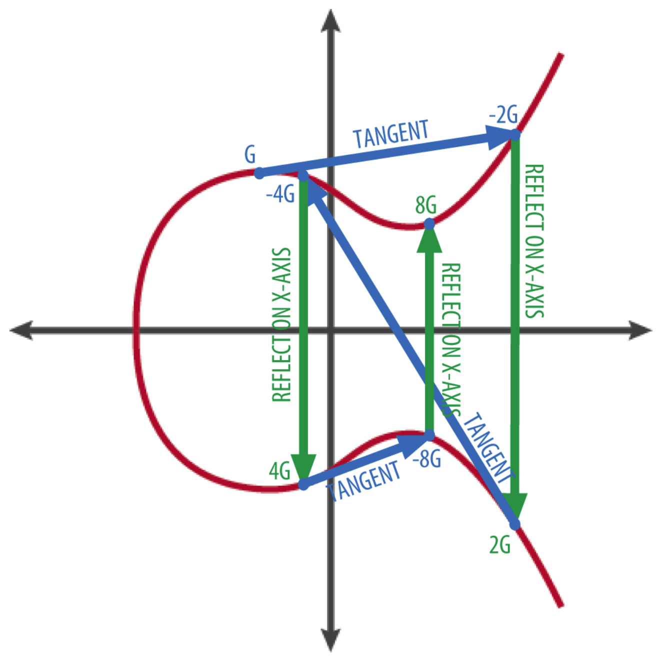
To better contextualise this curve to a use case within the Dusk Network, the bidding process can be used, as this connects several of the sections in this paper. The bidding process uses the arithmetic of the curve to perform operations, as well as the set inclusion principles to the properties of the bid. It is first necessary to show that a bid lies in the list of valid bids, i.e. is a subset of all valid bids. This is done by set membership to see if an element is part of the total set or by showing that the element is linear in N, where N is the size of the group. Then the necessary requirements for the bid are proven, which is making sure it hashes to the correct values. Following this, the bid is added to a vector of valid bids. A binary vector, which is a vector that compactly stores bits, is then created and this vector must be the same length as the vector of valid bids plus the created bid. In this binary format, a one is indicative of the position of your bid, and zero is indicative of the other bids.

6 Field Elements

For curve arithmetic to be performed, it is imperative to have a solid implementation. This allows for a basis on which the most primary operations can be carried out, the crucial nature of these operations stems from the ability to perform multiple cryptographic functions from only a few fundamental operations.

It is standard when implementing curves from their field elements, that point addition is the first function to be defined, as it is the foundation on which the rest of the operations stand. Point addition is simply adding points to one another along the elliptic curve.

The points which can be shown by x and y, in Cartesian form, lie upon the elliptic curve and are all multiples of the generator point. Setting the prime field, over which the curve is defined, aside for a moment allows for more clear mental imagery of how point addition works. The image below depicts point addition on a standard elliptic curve, with good visual aids. The generator point, denoted as G, is the point from which the addition is begun until the next generator point is reached. This is done by taking a tangent to the Generator point and then reflecting it on the x-axis, because of the mirror symmetry properties[7], which gives the next point. The image below provides the reader with a visual understanding of how the point addition can be performed:



Point addition varies from curve to curve and optimizations are continually performed whilst the field elements are created. The main rationale behind the need for optimization is to keep the operations time constant. The field elements are represented in bit terms, which are commonly converted to `u64` arrays. Unfortunately, the aforementioned formatting can lead to problems with the arithmetic in programming. These issues are often centred around over-spill, which occurs when making computations that have bit carrying. Such issues arises when using 32-byte arrays in addition, which impacts the overall performance as the operation leaves remainders due to the bit-carrying.

In order to avoid the issues mentioned above, radix representations of the field elements are utilized in order to avoid this bit-carrying as well as to eliminate any potential overflows created during addition, which makes the implementation more efficient. Every *field element* has to be represented as an array of five `u64`'s (in a concrete radix representation), which enables the computation of the product in the form $\texttt{u64} \cdot \texttt{u64} = \texttt{u128}$[1].

To achieve this, the chosen radix is 2^{52}, which is optimal for dealing with overspill. An issue which arises from the use of bit terms is the computational speed of the field arithmetic operations.

In this case, it is known that the most expensive CPU operation is the integer division. In order to avoid the operation highlighted above, an implementation all of the curve arithmetics is combined with bit-shifting techniques[8]. Bit-shifting is simply done by moving a series of bits to the left or right to achieve greater efficiency in a mathematical operation. When dealing with radices, there is always a need to add an integer so that the another module can be achieved, this integer is what is used for bit-shifting. The selection of this integer is a simple arithmetic operation on the defined prime of the field. If we let x be the remainder of the prime field, as shown below:

$$l = 2^{252} + x$$

The value of the integer x can be proven:

$$p = 0 \mod p$$

$$p = 2^{252} + x$$

$$2^{252} + x = 0 \mod p$$

$$2^{252} = -x \mod p$$

The integer x is then used in the calculations for radix 2^{52}, so that a different module can be achieved.

From this point addition, many of the further operations are made elementary as they all work with the manipulation of points, in some mathematical relation.

[1] Please note that the Corretto implementation is taking advantage of the Rust Programming Language support for 128-unsigned integer operations.

References

[1] Stanford University, University College London and BlockStream, Benedikt Bünz, Johnathan Bootle, Dan Boneh, Andrew Polestra, Pieter Wuille and Greg Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More.
https://eprint.iacr.org/2017/1066.pdf

[2] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In 17th Annual ACM Symposium on Theory of Computing (STOC'85), pages 291–304, 1985."

[3] Pedersen T.P. (1992) Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: Feigenbaum J. (eds) Advances in Cryptology — CRYPTO '91. CRYPTO 1991. Lecture Notes in Computer Science, vol 576. Springer, Berlin, Heidelberg

[4] Isis Lovecruft and Henry de Valence. Ristretto. https://Ristretto.group/Ristretto.html

[5] Mike Hamburg : Deacaf. November, 2015. https://eprint.iacr.org/2015/673.pdf

[6] Feng Hao, Thales E-Security, Cambridge, UK https://eprint.iacr.org/2010/149.pdf

[7] Robert Dijkgraaf: Mirror Symmetry and Elliptic Curves, university of Amsterdam, November 15, 2002

[8] Tehcnological University of Visvesvaraya, Jnana Sangama https://www.academia.edu/8777556/