Doppio Curve: Our use in a set inclusion cryptographic protocol.

Luke Pearson and Carlos Perez

***Abstract*— In this paper an explanation of the the current Doppio Curve development is given, as well as a contextual understanding of how this curve implementation acts as one of the aspects within the Dusk network.**

I. INTRODUCTION

In order to fully understand not only the theory but also the reasoning behind the use of such a curve, it is important that an understanding of our current goals is reached, so that the properties of the curve implementation and the choice for certain novel methods can be seen holistically and as part of a pragmatic solution.
One of our ultimate objectives, to put it bluntly, is to create set inclusion zero knowledge proofs using bulletproof methods on a Ristretto scalar field. To separate this into its fundamental parts, the first thing to analyse is set inclusion, which is the umbrella idea motivating the entire project.
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 were building.

II. SET INCLUSION

Set inclusion is the relationship of one set being a subset of another, in order for one to be a subset of another, the elements of the subset must exist in the set but cannot be the entire set.

A. Example

For a example if we have:

$$A = 1, 3, 5$$

$$i) B = 1, 5$$

then **B** is a subset, or ***proper subset of A.*** It is also important to note that: if

$$ii) B = 1, 3, 5$$

then **B would not be a subset of A as B=A**, in this case.
Also, if

$$iii) B = 1, 4$$

then **B would not be a subset of A**, as every element of a B must exist in A for the subset to exist

B. 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 isnt being used.
- Within curves, specifically elliptic curves, which is the central topic to this paper, the use of sets in divisor arithmetic allows us to show that the order of H is a divisor of the order of G, where H is a subgroup in a finite group G.

A full comprehension of this subset rule is very helpful, as well as largely applicable to our defined curve.
To contextualise the work on an aggregated level, the Dusk project has varying uses for set inclusion - one prominent case, is proving that our bid belongs in the bid set, which is all of the valid bids, without revealing the bid and then proving that the score calculation has been done correctly.
For our current set inclusion task, because the set elements are public keys and the input is a secret key, there needs to be a

$$ScalarBaseMult(P = xG) operation.$$

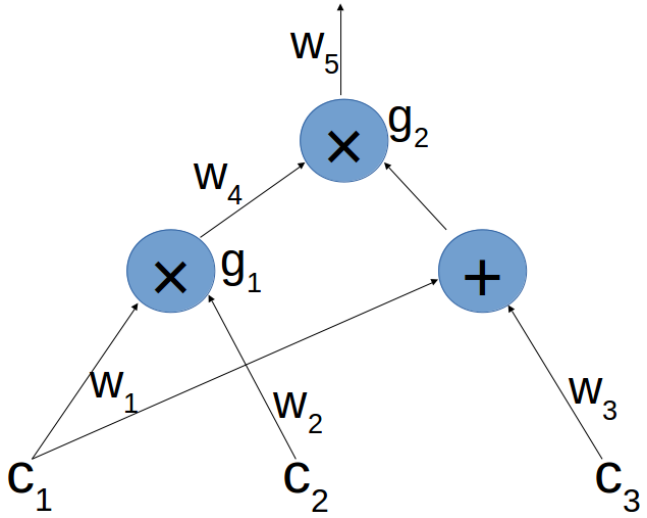
III. BULLETPROOFS AND RANK 1 CONSTRAINT CIRCUITS

Bulletproofs[1] are short non-interactive zero-knowledge proofs[2] that require no trusted setup. Bulletproofs can be used to convince a verifier that an encrypted plaintext is well formed. For example, they can be used to prove that an encrypted number is in a given range, without revealing anything else about the number. Compared to SNARKs[2], Bulletproofs require no trusted setup, which further reduces the risk of a malicious middle man. However, Bulletproofs verification is more time consuming than SNARK proof verification.

Bulletproofs are designed to enable efficient confidential transactions in Bitcoin and other cryptocurrencies. Every confidential transaction contains a cryptographic proof which proves the validity of the transaction. Bulletproofs shrink the size of the cryptographic proofs from over 10kB to less than 1kB. Moreover, Bulletproofs support proof aggregation, meaning that proving that m transaction values are valid adds only $O(log(m))$ additional elements to the size of a single proof.

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 optimisation to the *Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting* paper proposal, which showed that it was possible to prove arithmetic circuit satisfiability in a discrete log setting. This paper introduced a logarithmic-sized inner-product argument, as well as a technique to convert an arithmetic circuit into a rank-1 constraint system (R1CS). The Bulletproofs paper optimizes the inner-product argument by a constant factor, and shows how to use Pedersen commitments[3] as inputs to the constraint system. R1C's are the circuit format upon which Bulletproofs are built. The purpose for using these is that they expresses a linear combinations of various variables and their multiplications in a systematic manner.



The Bulletproofs paper uses the following format for the constraint system:

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

The downfall that Bulletproofs have is that they are not efficient with various elliptic curves and fields; although Bulletproof implementation provides a solid means of creating fast proofs, the prior choice of curve is important to ensure that the number of gates isn't too large thus effecting the time efficiency of the curve operations. As an example, a curve with fast and secure arithmetic operations such as the Curve25519[4] requires a large amount of gates to make use of a Bulletproof, because of the requirement to make a binary decomposition of every field element. The desire to avoid these constraint issues acutely summarises the reasoning behind the choice to implement a Doppio curve within the Ristretto Scalar Field[5]. The aforementioned curve simplifies the computational load (in terms of circuit gates), required to implement Bulletproofs on top of it.

IV. The Ristretto Scalar Field

The Ristretto scalar field is a defined field which constructs a prime-order group using a non prime-order Edwards curve. The Ristretto field builds upon the ideas of the Decaf paper[6], which is also constructs prime order curves but only those whose co-factor is equal to 4. Ristretto, on the other hand, is applicable to curves which have a co-factor equalling 8. The motive for this is to be able to avoid the drawbacks that are concurrent with a co-factor, whilst being able to capitalise on the robustness of an otherwise solid curve. In order for the reasoning behind this protocol to be shown, it is crucial that an understanding of co-factors and their potential pitfalls are given a contextual review.

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

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 exactly 8. There are various advantages and disadvantages to having a co-factor larger than one, which means the purpose of the implementation and chosen operations must be outlined in advance so that it is known whether co-factor manipulation is needed. In our case of set inclusion, having a co-factor larger than one will hinder our curve operations, specifically relating to the scalar base operations. In reference to our need for subset proofs, the co-factor pitfall which obstructs our goal is that where the co-factor is greater than one, there is non-invective behaviour between the groups. Non-injective functions in set mappings, which is just a relation that each element of a given set is shown with an element of another set, distorts the ability to prove subsets exists within sets. For Elliptic curves, any scalar multiplication is a 1 to 1 mapping if the scalar is relatively prime to the group order. The ability to choose a random scalar for the operation is from the range from 1 to q-1, is only valid in a prime-order group. Whereas in a non prime-order group, the adding of a small element can lead to 'identity misbinding', which makes it possible to present the same result from different inputs, hence nullifying a 1 to 1 operation. When the removal of these co-factor pitfalls is possible, a much larger selection of curves are available which can better suit the purpose of the overall implementation. From this, we can assume that we will have a curve of prime order that is constructed using the Ristretto scalar field.

To make use of this prime field, well be using the definition that for two curves L and M, L is embedded into M if the scalar field of M is equal to the field that L is

defined over. For this we will be using the Doppio curve, which will be eluded to shortly; this curve will be defined over the scalar Ristretto field. 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. Which as a method of creating parallels, is not dissimilar to how X, Y, and Z projective coordinates can represent the same P and Q Edwards points.

V. Equations

A. Twisted Edwards and Montgomery Forms

In order for a selected elliptic curve to be successful, it needs to be twist secure and Ristretto ready. From this we have chosen the Doppio curve.

Which is defined as follows:

- Curve equation

$$-x^2 + y^2 = 1 - \frac{86649}{86650}x^2y^2$$

  Which is Twisted Edwards.
- $Cofactor : h = 8$
- $a = -1$
- $d = \frac{86649}{86650}$
- $Basepoint : Y = \frac{8}{9}$

- Montgomery form equivalent:

$$y^2 = x^3 + 346598x^2 + x$$

- $A = 346598$
- $Basepoint : X = 17$

- The number of points on the curve is

  $2^{252} - 121160309657751286123858757838224683208$
- The prime order of the subgroup is

  $2^{249} - 151450387072189107654823447297780854 01$
- The prime order of the Ristretto scalar field is

  $2^{252} + 27742317777372353535851937790883648493$

B. Weierstrass Form

- Weierstrass form equivalent:

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

- a = 24123351924440874046577288543476647695237211979330253533398364605510802579 6
- b = 13401862180244930025876271413042581927513178443296125196299939987104848049 61

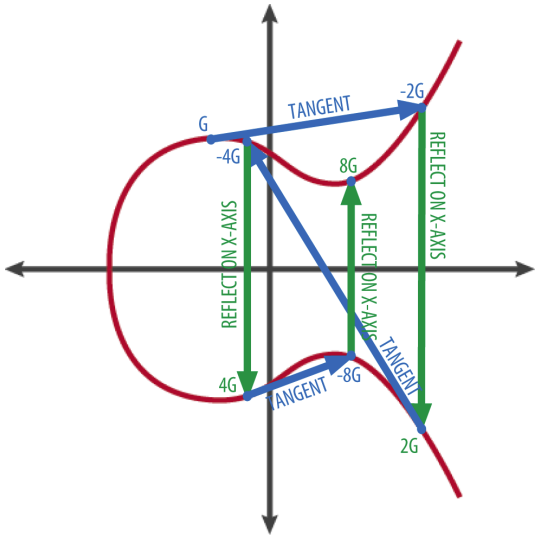
The computation for the Weierstrass form is to make point addition, and proving point addition is correct, as simplistic as possible because this underlines all of our current operations. These initial operation on the field elements are inline, which is to ensure they function as fast as possible.

VI. Field Elements

For successful 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 define, as it is the foundation on which the rest of the operations stand, as described in the previous paragraph. Point addition is simply adding points to one another along the elliptic curve.

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



Point addition varies from curve to curve and the optimisation is continually performed whilst the field elements are created. The main rationale behind the need for optimisation is that we will be keeping all of our operations time constant. The field elements are represented in bit terms, which are commonly converted to u64 arrays, this formatting can lead to problems with arithmetic in the programming. The problem is that working with 32-byte arrays in the addition is much less performant, as the operation leaves remainders due to the bit-carrying.

In order to avoid that, we use a Radix representations of the

Field Elements in order to avoid this bit-carrying as well as eliminating any overflowing on addition, which makes the overall code more performant. Every *Field Element* has to be represented as an array of five u64's (in a concrete Radix representation), which enables us to compute the product in the form $u64 * u64 = u128$.
Note that we are taking advantage of the Rust Programming Language, as it rudimentarily supports 128-unsigned integer operations.
To achieve this, we have decided to to use a Radix 2^{51}, 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 our case, we know that the most expensive CPU operation is the Integer Division. In order to avoid it, we've implemented all of the curve arithmetics with bit-shifting techniques[8]. Bit shifting is simply the moving of a series of bits to the left or right to achieve greater efficiency in a mathematical operation. When dealing with Radix, there is always a need to add an integer so that 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 from the prime field, as shown below.

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

We can prove the integer x and determine its value.

$$p = 0(modp)$$
$$p = 2^{252} + x$$
$$2^{252} + x = 0(modp)$$
$$2^{252} = -x(modp)$$

The integer x is then used in our calculations for radix 2^{51}, 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 manner or another.

REFERENCES

[1] Standford University, University College London and BlockStream, Benedikt Bnz, 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] Nir Bitansky , Ran Canetti , Alessandro Chiesa , Eran Tromer, From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again, Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, p.326-349, January 08-10, 2012, Cambridge, Massachusetts

[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] D. J. Bernstein. Curve25519: new Diffie-Hellman speed records. Proceedings of PKC 2006, to appear. Document ID: 4230efdfa673480fc079449d90f322c0. URL: http://cr.yp.to/papers.html#curve25519. Date: 2006.02.09.

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

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

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

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