Intuitive Interpretation of Non-Interactive Zero-Knowledge Cryptography

A jargon-free approach to understanding zk-SNARKs and zk-STARKs

Avneet Singh
Interplanetary Company UG
sshmatrix@proton.me

ABSTRACT

zk-SNARKs and zk-STARKs are relatively new concepts in cryptography, yet they are being touted as the next forefront in modern and future crypto tech. In blockchain space specifically, there is great interest in these subfields in the context of zk-Rollups to Layer 1 blockchains such as Ethereum, or as standalone decentralised ledgers with high rates of transactions per second (TPS), e.g. Aztec Network (zk-STARK), zkSync, Loopring, ZCash (zk-SNARKs) etc. Despite their great importance in cryptography, it is unfortunately difficult to understand zk-SNARKs and zk-STARKs due to limited literature and conceivably difficult mathematics conveyed through intensive jargon. This paper is an attempt to introduce zero-knowledge (zk) cryptography in an intuitive manner to garden-variety mathematicians, physicists, curious blockchain developers and perhaps even cryptographers.

INTRODUCTION

Zero-knowledge cryptography (ZKC) is presumably the next natural stage in cryptography's evolution toward post-quantum era. zk-SNARKs and zk-STARKs are specific implementations of zero-knowledge cryptography that are widely considered the most promising path toward post-quantum security. In order to deeply understand zero-knowledge cryptography, one must at least understand the current generation cryptographic systems such as RSA (Rivest–Shamir–Adleman) and Elliptic Curve Cryptography (ECC); this is admittingly a challenging task since the mathematics of such protocols is rather tedious and it only gets exponentially worse as one ventures in zero-knowledge protocols. Despite these challenges, it is nonetheless easier to understand at least the philosophy and intuition behind zero-knowledge protocols using the famous Alibaba Cave example [1] without requiring any prerequisite knowledge of RSA or ECC. We leave this as an exercise for the reader. In this paper, we will attempt to delve into the practical implementation of zero-knowledge protocols while retaining an intuitive understanding of the underlying mathematical processes.

> Through the course of this paper, additional comments in a grey box should be interpreted are anecdotes, comparisons or similariries that may help readers develop an intuitive understanding. In addition, we encourage readers to read this paper in conjunction with [10].

Cryptographic protocols at their core are motivated by the need to prove access to some information without necessarily revealing a part or the entirety of said information. In mathematical terms, there are several ways of achieving this functionality from an intelligently designed system.

PRIME NUMBERS

Cryptographers realised back in the day that prime numbers were one such system

that could provide the desired functionality. For instance, consider two sufficiently large prime numbers **a** = 53781811 and **b** = 23252729, and their even larger product **a** × **b** = 1250573876312219. The product 1250573876312219 is a relatively difficult number to prime factorise back to **a** and **b** if both **a** and **b** are unknown. However, if either one of the two prime factors (**a** or **b**) are known, then it is straightforward to calculate the other unknown prime factor by simple division. To intuitively understand this system further, let's break it down into its principle components: we took a set of very large prime numbers of which **a** and **b** are members and defined an operation of product[1] on the members of the set; such a finite field is called a Galois field. Additionally, we note that the product operation × is difficult to invert unless one of the two numbers is known; this kind of a system is usually called a trapdoor. Trigonometric functions like sine(**a**) in real space and e^{ia} in complex space are some other naive examples of trapdoors.

In abstract mathematics, such a system is called a Group and the study of groups is called Group theory; a group is defined by a set of elements (called a Field, e.g. large prime numbers) along with the set of permitted operations between those elements (e.g. product). The configuration of any trapdoor system is such that the permitted operations defined on the elements of the group are difficult to invert.

RSA CRYPTOGRAPHY

RSA protocol is one of the simplest implementations of the trapdoor feature which results in a keypair system – a public key and a private/secret key – typically utilised in conjunction for encrypting and decrypting information. The premise of the RSA protocol essentially lies in setting, i) public key as the qualitative equivalent of the product **a** × **b**, and ii) private key as the qualitative equivalent of either **a** or **b**, where **a** and **b** are restricted to a finite field of prime numbers. The security of such a system is encoded in the difficulty of prime factorising the product. It is unimportant to know the precise details of the protocol implementation in context of this paper. In nutshell, the RSA algorithm requires solving for the integer coefficients (x, y) of Bezout's Identity $x \cdot \mathbf{a} + y \cdot \mathbf{b} = 1$ using extended Euclidean algorithm [2] and employing modular arithmetic to wrap integer numbers when they fall outside the range of a separately defined composite finite field $F_{\mathbf{p}}$.

Modular arithmetic is quite simply equivalent to applying the modulo '%' operation over some prime number **p** to all group operations • (i.e. • = +, −, × or /), such that $\mathbf{a} \bullet \mathbf{b} \rightarrow \mathbf{a} \bullet \mathbf{b} \ \% \ \mathbf{p}$. Modular airthmetic is regular arithmetic 'wrapped' by the modulo operator such that results of all group operations lie within 0 and **p**; the range of % **p** is called a prime field $F_{\mathbf{p}}$. For example, new operations are: $\mathbf{a} + \mathbf{b} \rightarrow (\mathbf{a} + \mathbf{b}) \ \% \ \mathbf{p}$, $\mathbf{a} - \mathbf{b} \rightarrow (\mathbf{a} - \mathbf{b}) \ \% \ \mathbf{p}$, $\mathbf{a} \times \mathbf{b} \rightarrow (\mathbf{a} \times \mathbf{b}) \ \% \ \mathbf{p}$ and most notable, $\mathbf{a} \ / \ \mathbf{b} \rightarrow (\mathbf{a} \times \mathbf{b}^{\mathbf{p}-2}) \ \% \ \mathbf{p}$.

ROBUSTNESS OF RSA

While RSA is sufficiently safe to use today, it's safety will decrease over time as computational capacity of human civilisation increases. This is because the fastest and – arguably – maximally efficient algorithms capable of inverting the **a** × **b** product are iterative by construction and rely on optimised brute-forcing; Quadratic Sieve [3] and General Number Field Sieve [4] are two

[1] The operations of addition '+' and its inverse subtraction '−' are implictly defined

such well-known methods. With the advent of quantum computers, RSA algorithm's security will be definitively compromised; this is the so called SNDL problem (Save-Now-Decrypt-Later) facing the cryptography community today [5]. The core issue at hand here is that prime finite fields (meaning finite field of prime numbers; not to be confused with a prime field) and the generic operation of product on them – irrespective of the largeness of its elements – does not possess sufficient difficulty if the exploitor has relatively large computational resources at hand. In order to design better cryptosystems that are secure against brute-forcing, one must come with a new group with a better choice of finite field and a preferably a harder group operation which is resistant to such an attack.

DIFFIE-HELLMAN KEY EXCHANGE

In very few lines, let us discuss the Diffie-Hellman Key Sharing Algorithm. Unlike RSA which is a keypair-based algorithm with a public key identity, Diffie-Hellman is a 'secret sharing' method between two parties who each possess a secret private key and a shared public key. Diffie-Hellman in some sense is the intermediate step between RSA and ECC since it introduces relatively simple finite fields over groups other than itself; this simply means that instead of explicitly operating on a set of prime numbers (e.g. in RSA), we will instead operate on 'cyclic' functions of primes $\mathbf{C(p)}$.

> In a finite cyclic group with N elements (i.e. of "order" N), all N elements can be derived by repeatedly performing the group operation on one of the elements (called the generator). We can thus represent the K-th element in such a group with notation $\mathbf{G}^K$, representing K group operations on the generator element $\mathbf{G}$; note that the term $\mathbf{G}^K$ should not be interpreted as a literal exponent although a lot of literature uses this terminology. The use of exponential notation results from the classical Diffie-Hellman algorithm using the literal exponent of a 'primitive root modulo $\mathbf{p}$' $G = G^K$ as the group operation on prime finite fields, i.e. $\mathbf{G}^K \equiv (G \times G \times G \times G \times ... \times G)_{K = 0,1,2,3...(N-1)}$. The final form of $\mathbf{C(p)}$ thus reads $\mathbf{G}^K \ \% \ \mathbf{p}$. The cyclicity of e^{iz} in complex space is another reason for the exponential notation.

In practise, the private key for two parties is derived by evaluating the function $\mathbf{C(p)} = \mathbf{G}^K \ \% \ \mathbf{p}$ from commonly known values of $\mathbf{G}$ and $\mathbf{p}$ at two values of $K = \bar{\mathbf{p}}, \bar{\mathbf{q}}$. The shared key then takes the form of $\mathbf{G}^{\bar{\mathbf{p}}\bar{\mathbf{q}}} \ \% \ \mathbf{p}$, which is also the non-invertible term securing the protocol[2] instead of a simple product between two large prime numbers. Breaking the Diffie-Hellman method is thereby different from RSA since it involves inverting $\mathbf{G}^{\bar{\mathbf{p}}\bar{\mathbf{q}}} \ \% \ \mathbf{p}$ instead of $\mathbf{a} \times \mathbf{b}$; this is known as the Computational Diffie-Hellman Problem (C-DHP).

> More specifically, C-DHP states that given the knowledge of $\mathbf{G}^{\bar{\mathbf{p}}}$ and $\mathbf{G}^{\bar{\mathbf{q}}}$ (but not $\bar{\mathbf{p}}$ and $\bar{\mathbf{q}}$), it is difficult to compute the value of $\mathbf{G}^{\bar{\mathbf{p}}\bar{\mathbf{q}}}$. More simply, if we are revealed the values of two elements derived from secretly taking $\bar{\mathbf{p}}$ and $\bar{\mathbf{q}}$ cyclic operations, then it is difficult to compute the element derived by taking $\bar{\mathbf{p}} \times \bar{\mathbf{q}}$ cyclic operations. One variant of Diffie-Hellman problem which is easy to solve in this case is the Decisional Diffei-Hellman Problem (D-DHP). D-DHP states that given knowledge of three elements $\mathbf{G}^{\bar{\mathbf{p}}}$, $\mathbf{G}^{\bar{\mathbf{q}}}$ and $\mathbf{G}^{\bar{\mathbf{r}}}$ (but not $\bar{\mathbf{p}}$, $\bar{\mathbf{q}}$ or $\bar{\mathbf{r}}$), it is probabilistically easy to determine whether $\mathbf{G}^{\bar{\mathbf{p}}\bar{\mathbf{q}}} = \mathbf{G}^{\bar{\mathbf{r}}}$. In other words, given $\mathbf{G}^{\bar{\mathbf{p}}}$ and $\mathbf{G}^{\bar{\mathbf{q}}}$, while it is hard to determine $\mathbf{G}^{\bar{\mathbf{p}}\bar{\mathbf{q}}}$, it is relatively easier to

[2] Note that $\mathbf{a}'$ and $\mathbf{b}'$ are not necessarily prime numbers

determine whether a third randomly chosen element $G^{\bar{r}}$ is a result of $G^{\bar{p}\bar{q}}$. While the relative ease of solving D-DHP may not be obvious, the proof of it becomes apparent after taking into account the Law of Quadratic Reciprocity of Legendre symbols [9]; this however falls out of scope of this paper.

ELLIPTIC CURVE CRYPTOGRAPHY

The fundamental issue that leads to the breakdown of RSA algorithm is that the prime number field and the operation of product is not complex enough in the face of large computational power. The effort to improve on this problem led to the advent of Elliptic Curve Cryptography (ECC). ECC functions by fixing the core illnesses in RSA, i.e. it proposes – similar to Diffie-Hellman – that instead of operating on the prime number field itself, we operate on another field on which we define a newer set of operations that are much harder to invert. The premise of ECC thus lies in the introduction of prime finite fields over special elliptic curves, a step up from simple cyclic groups used in Diffie-Hellman [7]. In context of cryptography, an Elliptic curve is simply the following relation between a prime finite field $\mathbf{x}$ and composite field $\mathbf{y}$,

$$\mathbf{y}^2 = \mathbf{x}^3 + p \cdot \mathbf{x} + q,$$
$$\text{given } 4p^3 + 27q^2 \neq 0 \tag{1}$$

Gigabrains realised few decades ago that the elliptic curves have some very cool properties in context of invertibile binary operations. To begin with,

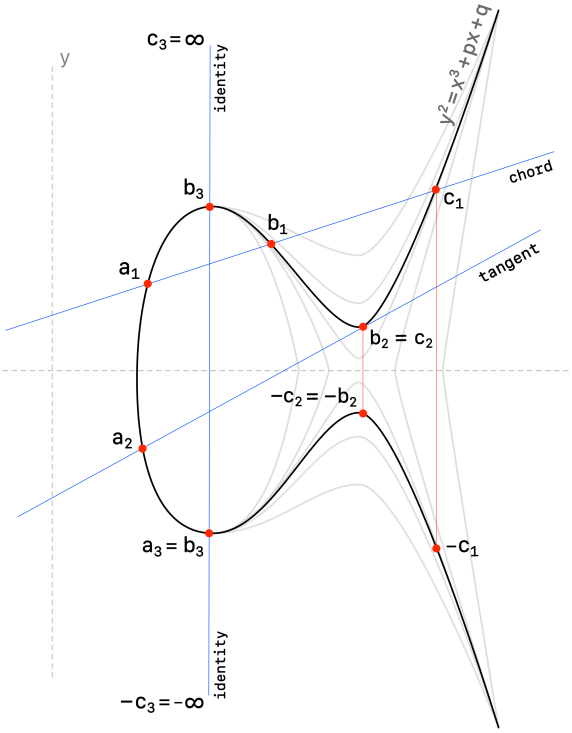

Figure 1: Elliptic curve over a continuous infinite field, aka a Lie group. We ignore the finite field contraint in this figure for simplicity. Three examples are shown depicting how to interpret different intersectionsof the line with the elliptic curve and the infinity point.

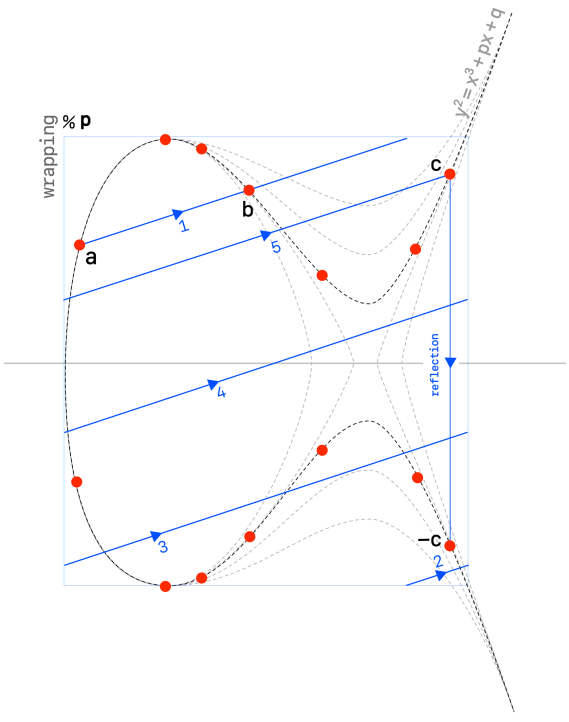

Figure 2: Example elliptic curve evaluation over a prime finite field modulo **p**; the effect of the modulo operator % is to "wrap" the out-of-range values to within range. In the above example, it takes 5 'wrappings' and 1 final reflection to calculate **−c**.

1. The elliptic curve is symmetric across the x-axis; this is desirable since it ensures existence of both **y** and **−y** in our group. More importantly,

2. For any two points **a** and **b** on the elliptic curve, a line through them intersects the elliptic curve again at most − but not necessarily − 1 more point, denoted by **c**; the blue line in figure 1 is one such example.

Typically, **x** is taken as the independent prime finite field and **y** is the so-called 'elliptic curve (evaluation) over finite field'; **y** are the elements of our desired new group. Ignoring the finite field constraint for a moment, the elliptic curve is shown in figure 1 when **x** is a continuous real variable. Our task now is to come up a hard-to-invert group operation on the continuous elements **y** of our new group; we will reinstate the prime finite field constraint afterwards. Note that it is possible for the line to not intersect the curve for the third time at all, and these singularities are in fact necessary for our group to function. Consider, the green line in figure 1; note that all vertical lines intersect the curve at exactly two points. It is also possible for a tangent to the curve to intersect the curve at exactly two points; see one such example (the red line) in figure 1. The aforementioned gigabrains soon realised that one can use this intersectional property of a line with the elliptic curve (aka properties 1. and 2.) as the new group operation $\oplus$ such that

$$\mathbf{a} \oplus \mathbf{b} = -\mathbf{c} \tag{2}$$

This is the same as saying that $\oplus$ operation between **a** and **b** is equivalent to taking reflection (−) of the third intersection point **c** across x-axis. Most literature phrases this $\oplus$ operation as the new "addition" operation on the group elements of **y**. However, we noted before that the third intersection **c** doesn't exist for all pairs **a** and **b**; in jargon-speak, we say that 'the group is not closed under the $\oplus$ operation'. We can solve this closeness problem by explicitely including the infinity element ∞ to the group. Once we have access to ∞, all intersections similar to the green line are assumed to intersect the curve again at $\mathbf{c} = \infty$, such that

$$\mathbf{a} \oplus \infty = \mathbf{a} \tag{3}$$

Note that ∞ acts like the identity element $\hat{I}$ of our new group since it satisfies $\mathbf{a} \oplus \hat{I} = \mathbf{a}$. Lastly, the third case of the red line is not difficult since it is simply a special case of the blue line when $\mathbf{b} = \mathbf{c}$; another important tangent case arises when $\mathbf{a} = \mathbf{b}$, such that $\mathbf{a} \oplus \mathbf{a} = \mathbf{c}$. Now we have all the ingredients required to construct a robust inverse problem on our new group **y** with the permitted operation $\oplus$. We reinstate the requirement of evaluating the elliptic curve on a prime finite field by explicitely defining a finite field range and wrapping the out-of-range evaluations over the elliptic curve with the modulo operator % **p**. The final form of the $\oplus$ operation is shown in figure 2. In a real world ECC implementation, this process is carried out a large number of times (assuming $\mathbf{a} = \mathbf{b}$, i.e. take tanget at point **a**) making the forward process increasingly complex but computable and the resulting inverse problem increasingly difficult to solve.

> In literature, you will often find the operation $\oplus$ equated to the generic addition $+$. This is not surprising since equations (2) and (3) resemble a + b = c and a + 0 = a for generic real numbers; 0 is the identity element of the $+$ operation. Having said that, in practice $\oplus$ operation is nothing like the addition operation $+$; the apparent similarity lies in their algebraic representations only. For instance, equations (2) and (3) also resemble $a \times b = c$ and $a \times 1 = a$ for generic real numbers. In this case, 1 is the identity element of the $\times$ operation.

The 'large number of repetitions' is assigned as the secret private key K_{sec} and the public key K_{pub} is equivalent to calculating $\mathbf{a} \oplus \mathbf{a}$ at a public value $\mathbf{a} = \mathbf{a}_o$ repeated K number of times[3], i.e. $K_{pub} = (...((\mathbf{a}_o \oplus \mathbf{a}_o)_1 \oplus \mathbf{a}_o)_2 \oplus \mathbf{a}_o)_3 ...)_K$. The exact inverse problem – called the Discrete Logarithm Problem (DLP) – is equivalent to finding the number of repetitions it took to arrive at K_{pub} from a publicly known value $\mathbf{a}_o$. Is this inverse problem hard? It appears that given knowledge of K_{pub} and $\mathbf{a}_o$, one can easily revert the process described in figure 2 and derive key K_{sec}. This is indeed true and the inverse problem only becomes hard when the coefficients in equation (1) are restricted to a finite field of large primes; more on this in the next subsection. Some of the popular elliptic curves today are specifically `ed25519` and `secp256k1`, with the later being more recent and more robust than the former.

> The name of the inverse problem for ECC, i.e. discrete logarithm problem (DLP), is highly misleading and results from purely notational reasons. The term 'logarithm' originates from interpreting the result of K times repetition of $\mathbf{a} \oplus \mathbf{a}$ operation such that $K_{pub} = \mathbf{a}^K$.

It now appears that we have achieved what we set out to do: we have found a new group who elements are now $\mathbf{y}$ in prime field F_p evaluated over the prime finite field and the new group operation $(...((\mathbf{a}_o \oplus \mathbf{a}_o)_1 \oplus \mathbf{a}_o)_2 \oplus \mathbf{a}_o)_3 ...)_K \equiv \mathbf{a}_o{}^K$ is very difficult to invert. We note that the result of $\oplus$ operation on the group elements results in a value which is also a member of the group[4]; this was not the case in RSA scheme where the result $\mathbf{a} \times \mathbf{b}$ is never a member of the group.

Curious observers may note that unlike RSA where we keep both $\mathbf{a}$ and $\mathbf{b}$ secret (and reveal their product $\mathbf{a} \times \mathbf{b}$), in ECC we additionally also reveal one of the independent variables $\mathbf{a}_o$, i.e. $\mathbf{a}_o{}^K$ and $\mathbf{a}_o$ are both known, leaving only K to be discovered. Why is this the case? The answer lies in the fact that the solutions $\mathbf{a}$ and $\mathbf{b}$ of the product $\mathbf{a} \times \mathbf{b}$ are unique while the solutions $\mathbf{a}$ and K of the expression $\mathbf{a}^K$ are not unique unless either $\mathbf{a}$ or K is specified. Uniqueness of the keypairs is a very important property that cannot be violated for cryptosystems; this requires $\mathbf{a}_o$ to be fixed and specified publicly in ECC algorithms.

Curious observers may additionally note that the expression $\mathbf{a}^K$ looks very similar to the $\mathbf{G}^K$ expression in Diffie–Hellman. How similar are these two? Quite a lot actually. The group defined by $\mathbf{a}^K$ is in fact also a finite cyclic group with $\mathbf{a}_o$ as its generator element! The order N, i.e. number of elements in the group, is defined by the expression $\mathbf{a}_o{}^N = \hat{1} = \infty$. Does this mean we can construct an even safer Diffie–Hellman Key Sharing algorithm by using an elliptic curve to construct the cyclic field instead of the simple choice made in classical Diffie–Hellman? The answer is yes we can and such methods already exist in the form of Elliptic Curve Diffie–Hellman algorithms [8].

ROBUSTNESS OF ECC

The natural question to ask here is: how robust is ECC when compared to RSA and Diffie–Hellman? We remember from previous section that the inverse problem in ECC only becomes hard when the coefficients p and q in equation (1) are also restricted to a finite field of large primes. To understand why this happens, simply note that for large prime values of $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{x}$ in equation (1), all

[3] $\mathbf{a} = \mathbf{b}$ assertion is made purely for simplicity since having $\mathbf{a} \neq \mathbf{b}$ doesn't provide extra security and in fact makes the forward computation unnecessarily require two generator elements.
[4] The group is referred to as 'closed under the operation $\oplus$'

terms on the right side are products of primes and then their sums. Similar to RSA, inverting this function numerically is equivalent to prime factorising many $\mathbf{a} \times \mathbf{b}$ products (and 'inverting' their sums). The fact that this process will have to be repeated a very large number of times (at least K_{sec} times) to potentially bruteforce for the private key, makes the ECC algorithm several orders of magnitudes more secure than RSA or Diffie-Hellman. While this is certainly an improvement, the realisation must dawn on us that the ECC is not a new flavour of cryptography but essentially an RSA-variant on steroids, in the sense that it 'supercharges' the prime factorisation problem for large prime products by redefining the group and its native operation $\oplus$. The increase in complexity in ECC lies entirely in the redefined native operation $\oplus$ which is a supercharged mix of addition $+$ and product $\times$ among large prime or composite numbers. Breaking ECC is therefore equivalent to breaking a very large number of RSA-like atomic prime factorisations. While the incremental complexity of ECC over RSA is sufficient at the moment, it will certainly not be sufficient in the post-quantum era; this finally brings us at the edge of the world of zero-knowledge cryptography. But before we step in, we need to discuss one final extension of ECC which is a precursor to Zero-Knowledge Cryptography (ZKC).

PAIRING-BASED CRYPTOGRAPHY

Pairing-based cryptography (PBC) is an extension of ECC which historically stems from Diffie-Hellman implementation of ECC and forms a crucial prelude to ZKC. PBC is around the point in literature when people start to lose their mind; most of this is a result of notational abuse and use of unintuitive terms such as r-torsion and divisor subgroups. The core premise of PBC is not very far from ECC and easily understood without losing our collective minds. In this paper, we will try to understand pairing-based elliptic curve cryptography using the example of a naive but intuitive cyclic group: e^{ia} in the complex space, i.e. $i^2 + 1 = 0$.

The vector notation $e^{i\alpha}$ of this group is the usual complex number $\sin(\alpha) + i \cdot \cos(\alpha)$, which is simply a vector in complex space inclined at an angle $\alpha$with the x-axis. Varying the value of αin this geometric representation is equivalent to rotating the vector by some angle about the origin. Quite naturally, any resulting finite group from $e^{i \cdot k\alpha}$ is cyclic for any generator value of α since all of its elements can be derived by simply by varying k, i.e. rotating the generator vector incrementely by α.

This is a good time to make some additional observations about finite cyclic groups using the simple example of $e^{i \cdot k\alpha}$, equivalent to $e^{k \cdot z}$ where $\mathbf{z}$ is a complex number. The resulting group is then simply vectors in complex plane which rotate (say, counter-clockwise) around the origin in increments of α as k varies. If α is a factor of 2π, then any vector starting from $k = 0$ will coincide with itself after precisely $N = 2\pi/\alpha$ turns. In other words, if we were to take any starting vector $\mathbf{z}(\alpha_0)$ as the identity element, then the resulting group is cyclic with an order $N = 2\pi/\alpha$; all of its elements can be derived by simply rotating the generator vector $e^{i \cdot \alpha_0}$ in increments of α up to N times.

Pairing-based cryptography is entirely reliant

RECAP

Before moving forward, let's recap the process of our rediscovery of cryptography. Cryptosystems are motivated by the need to prove access to

information without revealing a part or the entirety of the said information; such systems form the building blocks of countless cryptographic methods such as encryption/decryption, permissionless access, secure key sharing, digital signatures, blockchains etc. Such information-preserving proving algorithms are called 'trapdoors' since such algorithms – similar to a trapdoor – are easy to compute in one direction but difficult to invert. RSA is one of the simplest implementations of such trapdoors and it relies on the (non-)invertibility of product operation $\mathbf{a} \times \mathbf{b}$ on a prime finite field; this securing property is called the prime factorisation problem. RSA algorithm is relatively safe to use at the moment but it is expected that it will be broken within the next decade. ECC is the next generation of cryptographic trapdoor algorithms which essentially supercharges the prime factorisation problem by several orders of magnitude. It does so by resampling the group elements from an elliptic curve evaluated over a prime finite field and defining a new operation $\oplus$ on this cyclic group which captures the supercharging effect composed of large number of atomic prime factorisations at its core. The security of ECC relies on the difficulty of discrete logarithm problem, i.e. how many $\oplus$ group operations on the generator element $\mathbf{a}_o$ does it take to arrive at a point $\mathbf{a}_o{}^K$ on the elliptic curve. The supercharging increases the security of the resulting trapdoor but fails to increase it beyond the post-quantum regime due to being limited by the atomic difficulty of prime factorisation problem.

While the difficulty of ECC can be further channeled by choosing specific elliptic curves, gigabrains realised couple of decades ago that choosing a different ECC doesn't resolve the fundamental issue that plagues ECC. At the very core, solving discrete logarithm problem for (i.e. calculate K given $\mathbf{a}_o{}^K$) is bruteforceable since it reveals too much information in the form of generator element $\mathbf{a}_o$; this is necessary for the uniqueness of keypairs but it unwittingly reveals information that helps post-quantum bruteforce algorithms. Is this avoidable in any way? It doesn't appear so since knowing $\mathbf{a}_o$ publicly is necessary for deriving public keys by independent parties. It appears that to truly make progress, we may have to abandon our primary use of elliptic curves. Enter zero-knowledge cryptography.

ZERO-KNOWLEDGE CRYPTOGRAPHY

REFERENCES

[1] Quisquater, JJ. et al. (1990), How to Explain Zero-Knowledge Protocols to Your Children, Lecture Notes in Computer Science, Vol. 435, Springer, New York, NY
[2] Extended Euclidean Algorithm
[3] Quadratic Sieve Algorithm
[4] General Number Field Sieve Algorithm
[5] SNDL Problem
[6] Diffie-Hellman Key Exchange
[7] Elliptic Curve Cryptography
[8] Elliptic Curve Diffie-Hellman Algorithm
[9] Decisional Diffie-Hellman Problem
[10] Pairing-Based Cryptography; Craig Costello
[11] –

METADATA

Github:
Contracts:
Source:
SHA-1 Checksum:
Date: August 4, 2023