Maths Primer for Zero Knowledge

Terminology

Numbers

The set of Integers is denoted by $\mathbb{Z}$ e.g. {⋯,−4,−3,−2,−1,0,1,2,3,4,⋯}
The set of Rational Numbers is denoted by $\mathbb{Q}$ e.g. $\{\ldots 1, \frac{3}{2}, 2, \frac{22}{7} \ldots\}$
The set of Real Numbers is denoted by $\mathbb{R}$ e.g. $\{2, -4, 613, \pi, \sqrt{2}, \ldots\}$

Fields are denoted by $\mathbb{F}$, if they are a finite field or $\mathbb{K}$ for a field of real or complex numbers we also use $\mathbb{Z}_p^*$ to represent a finite field of integers mod prime p with multiplicative inverses, these concepts will be explained further later.

We use finite fields for cryptography, because elements have “short”, exact representations.

Modular Arithmetic

When we write n mod k we mean simply the remainder when n is divided by k. Thus
25 mod 3 = 1,
15 mod 4 = 3,
−13 mod 5 = -3 , = 2 mod 5.
It is an important fact that modular arithmetic respects sums and products.
That is,
a+b mod n = a mod n+ b mod n
and
a·b mod n=(a mod n)·(b mod n)

Group Theory

Simply put a group is a set of elements {a,b,c,…} plus a binary operation, here we represent this as •
To be considered a group this combination needs to have certain properties

1. Closure
   For all a, b in G, the result of the operation, a • b, is also in G
2. Associativity
   For all a, b and c in G, (a • b) • c = a • (b • c)
3. Identity element
   There exists an element e in G such that, for every element a in G, the equation e • a =

a • e = a holds. Such an element is unique and thus one speaks of the identity element.

4. Inverse element
   For each a in G, there exists an element b in G, commonly denoted a^{-1} (or −a, if the operation is denoted “+”), such that a • b = b • a = e, where e is the identity element.

Fields

A field is a set of say Integers together with two operations called addition and multiplication. One example of a field is the Real Numbers under addition and multiplication, another is a set of Integers mod a prime number with addition and multiplication.
The field operations are required to satisfy the following field axioms. In these axioms, a, b and c are arbitrary elements of the field $\mathbb{F}$.

1. Associativity of addition and multiplication: a + (b + c) = (a + b) + c and a · (b · c) = (a · b) · c.
2. Commutativity of addition and multiplication: a + b = b + a and a · b = b · a.
3. Additive and multiplicative identity: there exist two different elements 0 and 1 in $\mathbb{F}$ such that a + 0 = a and a · 1 = a.
4. Additive inverses: for every a in F, there exists an element in F, denoted −a, called the additive inverse of a, such that a + (−a) = 0.
5. Multiplicative inverses: for every a ≠ 0 in F, there exists an element in F, denoted by a^{-1}, called the multiplicative inverse of a, such that a ·a^{-1} = 1.
6. Distributivity of multiplication over addition: a · (b + c) = (a · b) + (a · c).

To try out operations on finite fields, see https://asecuritysite.com/encryption/finite (https://asecuritysite.com/encryption/finite)

For a great introduction see http://coders-errand.com/zk-snarks-and-their-algebraic-structure/ (http://coders-errand.com/zk-snarks-and-their-algebraic-structure/)

The **order** of the field is the number of elements in the field’s set.
For a finite field the order must be either

- prime (a prime field)
  or
- the power of a prime (an extension field)

An element can be represented as an integer greater or equal than 0 and less than the field’s order: {0, 1, …, p-1} in a simple field.

In a finite field of order q, the polynomial $X^q - X$ has all q elements of the finite field as roots.

Group Homomorphisms

A homomorphism is a map between two algebraic structures of the same type, that preserves the operations of the structures.

This means a map $f : A \to B$ between two groups A, B equipped with the same structure such that,

if $\cdot$ is an operation of the structure (here a binary operation), then
$f(x \cdot y) = f(x) \cdot f(y)$

Polynomials

A polynomial is an expression that can be built from constants and variables by means of addition, multiplication and exponentiation to a non-negative integer power.

e.g. $3x^2 + 4x + 3$

Lagrange Interpolation

If you have a set of points then doing a Lagrange interpolation on those points gives you a polynomial that passes through all of those points.

If you have two points on a plane, you can define a single straight line that passes through both, for 3 points, a single 2nd-degree curve (e.g. $5x^2 + 2x + 1$) will go through them etc. For n points, you can create a n-1 degree polynomial that will go through all of the points.

Adding, multiplying and dividing polynomials

We can add, multiply and divide polynomials, we don't need to go onto the details here, for examples see https://en.wikipedia.org/wiki/Polynomial_arithmetic
(https://en.wikipedia.org/wiki/Polynomial_arithmetic)

For a polynomial P of a single variable x in a field K and with coefficients in that field, the root r of P is an element of K such that $P(r) = 0$

B is said to divide another polynomial A when the latter can be written as

$A = BC$

with C also a polynomial,the fact that B divides A is denoted $B|A$

If one root r of a polynomial $P(x)$ of degree n is known then polynomial long division can be used to factor $P(x)$ into the form
$(x - r)(Q(x))$
where

$Q(x)$ is a polynomial of degree $n-1$.
$Q(x)$ is simply the quotient obtained from the division process; since r is known to be a root of $P(x)$, it is known that the remainder must be zero.

Schwartz-Zippel Lemma stating that “different polynomials are different at most points”.

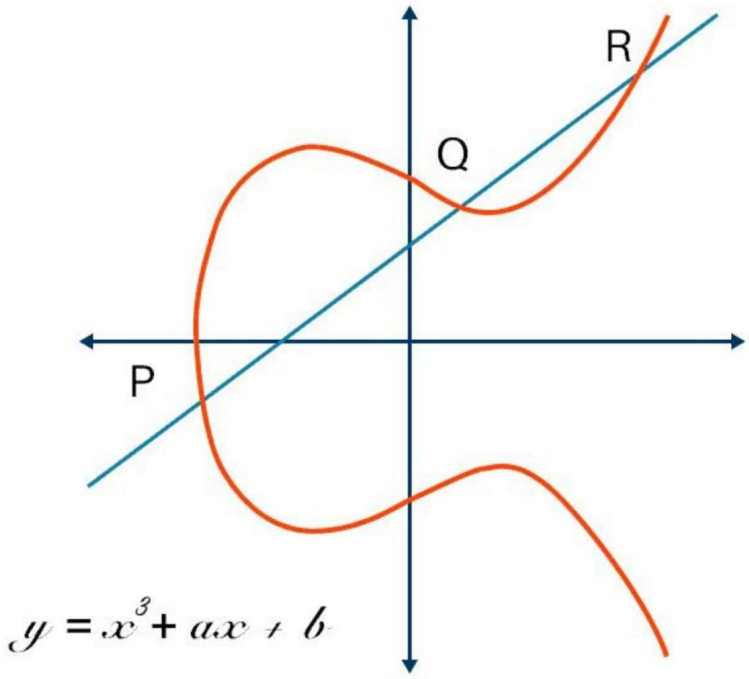
Elliptic Curves

The defining equation for an elliptic curve is for example $y^2 = x^3 + ax + b$
For certain equations they will satisfy the group axioms

- every two points can be added to give a third point (closure);
- it does not matter in what order the two points are added (commutatitivity);
- if you have more than two points to add, it does not matter which ones you add first either (associativity);
- there is an identity element.

We often use 2 families of curves :

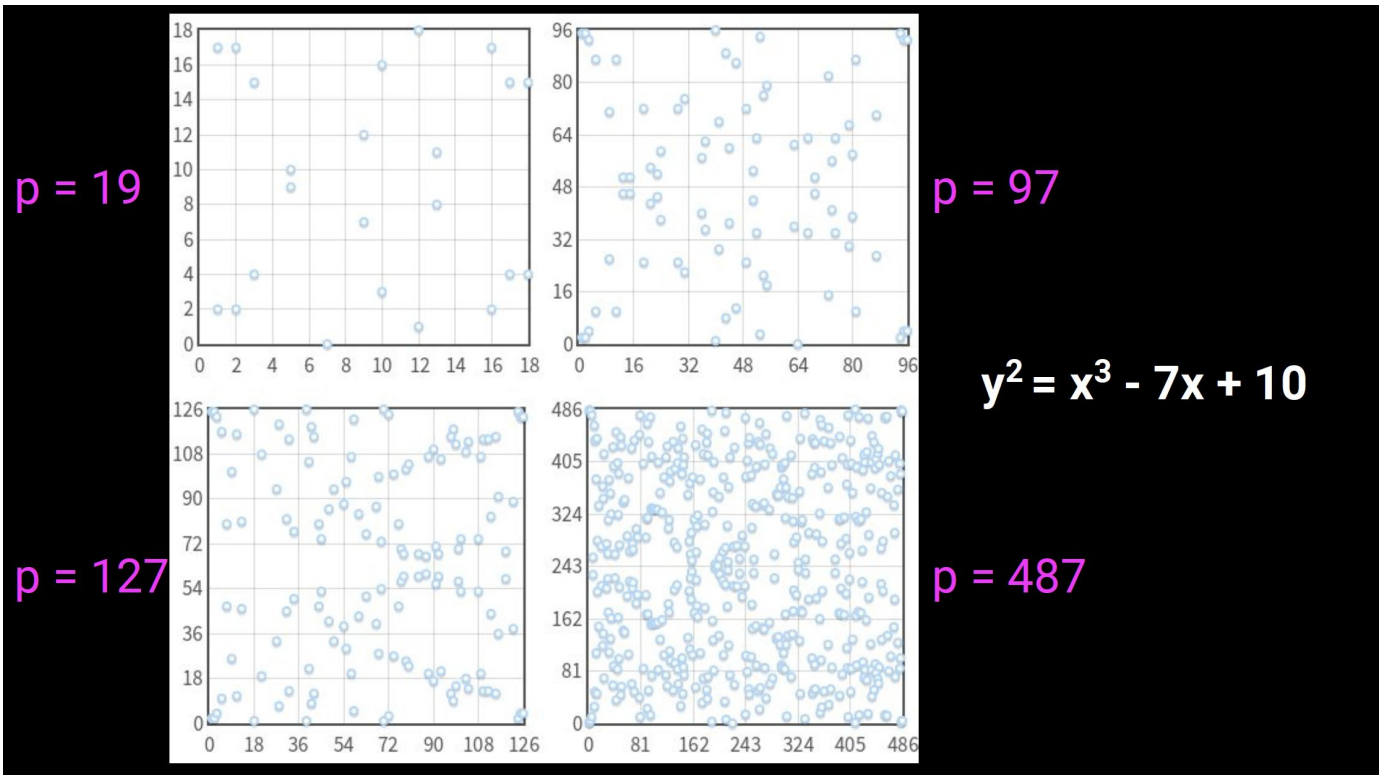
Montgomery Curves



For example curve 22519 with equation $y^2 = x^3 + 486662x^2 + x$
Generally this curve is considered over a finite field $\mathbb{K}$ (with order different from 2)

Curve 25519 gives 128 bits of security and is used in the Diffie–Hellman (ECDH) key agreement scheme
BN254 / BN_128 is the curve used in Ethereum for ZKSNARKS

BLS12-381 is the curve used by ZCash



Edwards Curves

The general equation is $ax^2 + y^2 = 1 + dx^2y^2$ with a = 1

If a <> 1 they are called Twisted Edwards Curves
Every twisted Edwards curve is birationally equivalent to a Montgomery curve

Pairings

Bilinear pairings are functions that take two arguments and return one output, usually denoted by

```
e(G1, G2) --> GT.
```

with the following properties:

- Order:All three groups must have order equal to a prime r.
- Efficiency: the pairing function must be efficiently computable.
- Bilinearity: For any elements P1, P2 of G1 and any elements Q1, Q2 of G2, the following holds true:

```
e(P1 + P2, Q1) = e(P1, Q1) e(P2, Q1)
e(P1, Q1 + Q2) = e(P1, Q1) e(P1, Q2)
```

This implies the following form, which is more often used (along with some other variants):

```
e(aP, bQ) = e(P,Q)ab = e(bP, aQ)
```

- Non-degeneracy: the pairing of the generators of the first two groups is not the identity of the third group. If this were the case, every pairing would result in the same (the identity) element of GT:

  $e(E1, E2) \neq 1T$

G2 is an elliptic curve, where points satisfy the same equation as G1, except where the coordinates are elements of $F_{p^{12}}$ (this is an extension field, where the elements of the field are polynomials of degree 12)
GT is the type of object that the result of the elliptic curve goes into. In the curves that we look at, GT is also $F_{p^{12}}$

Homomorphic Encryption

Homomorphic encryption is a form of encryption with an additional evaluation capability for computing over encrypted data without access to the secret key. The result of such a computation remains encrypted. Homomorphic encryption can be viewed as an extension of either symmetric-key or public-key cryptography. Homomorphic refers to homomorphism in algebra: the encryption and decryption functions can be thought as homomorphisms between plaintext and ciphertext spaces.

Bitcoin split-key vanity mining

Bitcoin addresses are hashes of public keys from ECDSA key pairs. A vanity address is an address generated from parameters such that the resultant hash contains a human-readable string (e.g., 1BoatSLRHtKNngkdXEeobR76b53LETtpyT). Given that ECDSA key pairs have homomorphic properties for addition and multiplication, one can outsource the generation of a vanity address without having the generator know the full private key for this address.

For example,
Alice generates a private key (a) and public key (A) pair, and publicly posts A.
Bob generates a key pair (b, B) such that hash(A + B) results in a desired vanity address. He sells b and B to Alice.
A, B, and b are publicly known, so one can verify that the address = hash(A + B) as desired.
Alice computes the combined private key (a + b) and uses it as the private key for the public key (A + B).
Similarly, multiplication could be used instead of addition.

Homomorphic Hiding

(Taken from the ZCash explanation (https://electriccoin.co/blog/snark-explain/))

If $E(x)$ is a function with the following properties

- Given $E(x)$ it is hard to find x
- Different inputs lead to different outputs so if $x \neq y$ $E(x) \neq E(y)$
- We can compute $E(x+y)$ given $E(x)$ and $E(y)$

The group $\mathbb{Z}_p^*$ with operations addition and multiplication allows this.

Complexity Theory

Complexity theory looks at the time or space requirements to solve a problem, particularly in terms of the size of the input.
We can classify problems according to the time required to find a solution, for some problems there may exist an algorithm to find a solution in a reasonable time, whereas for other problems we may not know of such an algorithm, and may have to 'brute force' a solution, trying out all potential solutions until one is found.

For example the travelling salesman problem tries to find the shortest route for a salesman required to travel between a number of cities, visiting every city exactly once. For a small number of cities, say 3, we can quickly try all alternatives to find the shortest route, however as the number of cities grows, this quickly becomes unfeasible.

Based on the size of the input n , we classify problems according to how the time required to find a solution grows with n.
If the time taken in the worst case grows as a polynomial of n, that is roughly proportional to n^k for some value k, we put these problems in class P. These problems are seen as tractable.

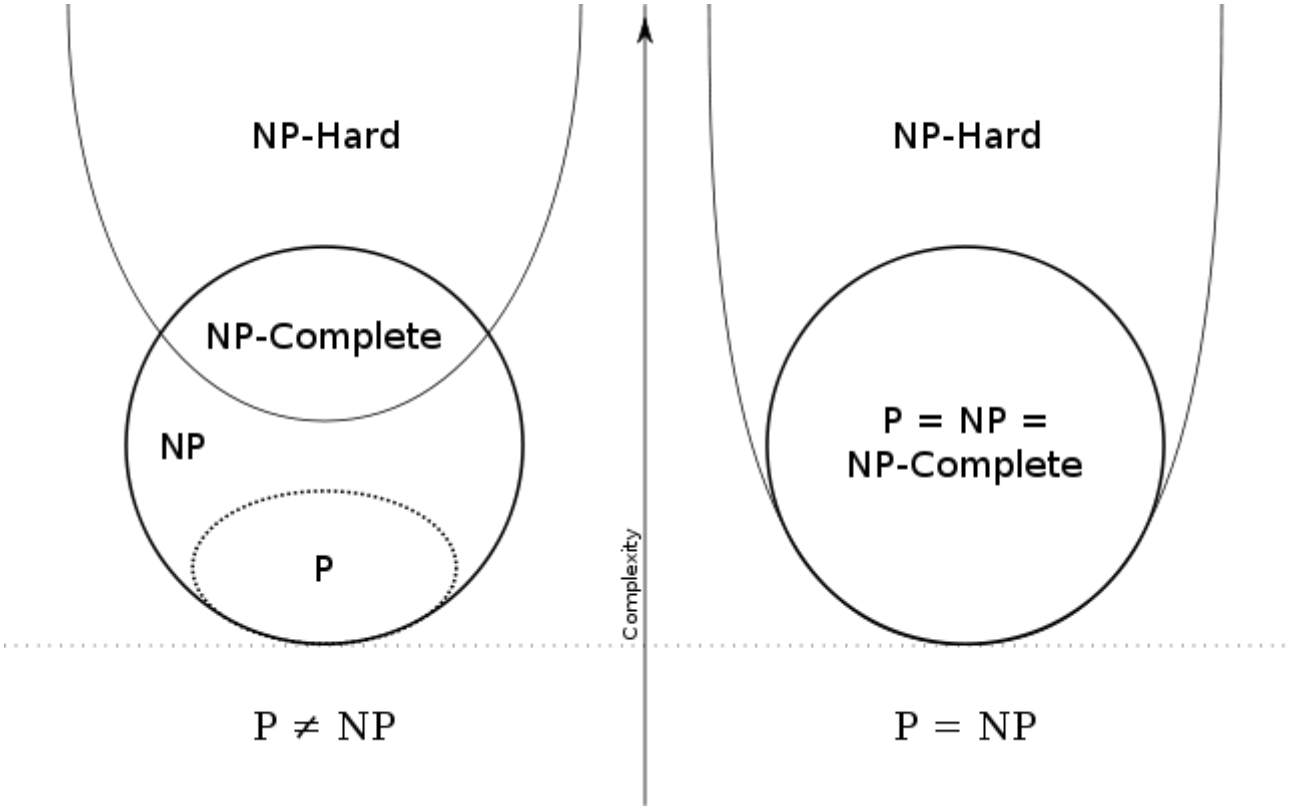
We are also interested in knowing how long it takes to verify a potential solution once it has been found.

A computational problem can be viewed as an infinite collection of instances together with a solution for every instance. The input string for a computational problem is referred to as a problem instance, and should not be confused with the problem itself.

https://en.wikipedia.org/wiki/Computational_complexity_theory
(https://en.wikipedia.org/wiki/Computational_complexity_theory)

Decision Problem: A problem with a yes or no answer

Complexity Classes



P

P is a complexity class that represents the set of all decision problems that can be solved in polynomial time. That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

NP

NP is a complexity class that represents the set of all decision problems for which the instances where the answer is “yes” have proofs that can be verified in polynomial time. This means that if someone gives us an instance of the problem and a witness to the answer being yes, we can check that it is correct in polynomial time.

NP-Complete

NP-Complete is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time.
Intuitively this means that we can solve Y quickly if we know how to solve X quickly.
Precisely, Y is reducible to X, if there is a polynomial time algorithm f to transform instances y of Y to instances
x = f(y) of X in polynomial time, with the property that the answer to y is yes, if and only if the answer to f(y) is yes

NP-hard

Intuitively, these are the problems that are at least as hard as the NP-complete problems. Note that NP-hard problems do not have to be in NP, and they do not have to be decision problems.
The precise definition here is that a problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.
But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.

Commitment schemes

Definition A commitment scheme is defined by algorithms **Commit** and **Open** as follows:

Given a message m and randomness r, compute as output a value c

```
c = Commit(m,r).
```

that, informally, hides message m and r such that it is hard to compute message m' and randomness r' that satisfies
```
Commit(m',r') = Commit(m r).
```
In particular, it is hard to invert function Commit to find
m or r.

Given a commitment c, a message m and randomness r
```
b = Open(c, m, r).
```
the algorithm returns true if and only if
c = Commit(m, r).
A commitment scheme has 2 properties:

1. Binding. Given a commitment c, it is hard to compute a different pair of message and randomness whose commitment is c. This property guarantees that there is no ambiguity in the commitment scheme, and thus after c is published it is hard to open it to a different value.
2. Hiding. It is hard to compute any information about m given c.

Pedersen commitments

You could hash the amount of transaction to hide it but that would be suseptible to rainbow table attacks.
So we add a blinding factor

Com(v) = vG + bH
Where G and H are generator points
b is random number used as a blinding factor

- Given group $\mathbb{Z}_p^*$, of prime order p, where the discrete logarithm problem is infeasible, the commitment is computed for message m and randomness r as follows:
  `c = Commit(m,r)`
  In order to open this commitment, given message m and randomness r, we simply recompute it and compare with c.
  An interesting property is that the Pedersen commitment is homomorphic.

```
C(BF1, data1) + C(BF2, data2) == C(BF1 + BF2, data1 + data2)
C(BF1, data1) - C(BF1, data1) == 0
```

Pedersen commitments are information-theoretically private: for any commitment you see, there exists some blinding factor which would make any amount match the commitment.

Fiat–Shamir heuristic

The Fiat–Shamir heuristic is a technique in cryptography for taking an interactive proof of knowledge and creating a digital signature based on it.

Here is an interactive proof of knowledge of a discrete logarithm.

1. Peggy wants to prove to Victor the verifier that she knows x: the discrete logarithm of $y = g^x$ to the base g
2. She picks a random $v \in \mathbb{Z}_q^*$ computes $t = g^v$ and sends t to Victor.
3. Victor picks a random $c \in \mathbb{Z}_q^*$ and sends it to Peggy.
4. Peggy computes $r = v - cx$ and returns r to Victor.

He checks whether $t \equiv g^r y^c$.
This holds because $g^r y^c = g^{v-cx} g^{xc} = g^v = t$

The Fiat–Shamir heuristic allows us to replace the interactive step 3 with a non-interactive random oracle access. In practice, we can use a cryptographic hash function instead.

1. Peggy wants to prove to Victor the verifier that she knows x: the discrete logarithm of $y = g^x$ to the base g
2. She picks a random $v \in \mathbb{Z}_q^*$ computes $t = g^v$
3. Peggy computes $c = H(g, y, t)$ where $H()$ is a cryptographic hash function.
4. She computes $r = v - cx$. The resulting proof is the pair (t, r). As r is an exponent of g, it is calculated modulo $q - 1$, not modulo q.
5. Anyone can check whether $t \equiv g^r y^c$.

Schnorr Protocol / Sigma Protocols

For a cyclic group G_q of order q with generator g.
In order to prove knowledge of $x = \log_g y$, the prover interacts with the verifier as follows:

In the first round the prover commits himself to randomness r therefore the first message $t = g^r$ is also called commitment.
The verifier replies with a challenge c chosen at random.
After receiving c, the prover sends the third and last message (the response) $s = r + cx$
The verifier accepts, if $g^s = ty^c$
Protocols which have the above three-move structure (commitment, challenge and response) are called sigma protocols

Sigma protocols can be used to prove any primitive that can be expressed as operations on group exponentiations.

BLS Signature Scheme

A scheme based on pairing cryptography consisting of 3 functions

1. Key generation
   The key generation algorithm selects a random integer x, the private key in the interval $[0, r-1]$. The public key is g^x.

2. Signing
   Given the private key x, and some message m, we compute the signature by hashing the bitstring m, as $h = H(m)$.
   We output the signature $\sigma = h^x$

3. Verification
   Given a signature σ and a public key g^x we verify that
   $e(\sigma, g)$=$e(H(m), g^x)$.

Ring Signatures

This is a type of digital signature involving a group of users that each have keys.
A transaction signed with a ring signature is endorsed by someone in a particular group of people.
It should be computationally infeasible to determine which of the group members' keys was used to produce the transaction signature.

Group Signatures

See Overview (https://pdfs.semanticscholar.org/97e8/d3830863b96a47f187d093981b4e37f5e2f8.pdf)

Roles

- Group Manager
- Group Member
- Verifier

Properties

- Unforgeability
- Unlinkability
- No framing
- Soundness
- Completeness

Keys

1. Master public key
2. Master secret key
3. Administrative key

Blind Signatures

The content of a message is disguised (blinded) before it is signed.
Intuitively an analogy is a voter signing a ballot in a carbon paper lined envelope.
An official verifies the credentials and signs the envelope, thereby transferring their signature to the ballot inside via the carbon paper.
Once signed, the package is given back to the voter, who transfers the now signed ballot to a new unmarked normal envelope.
Thus, official signing does not view the message content, but a third party can later verify the signature and know that the signature is valid within the limitations of the underlying signature scheme.

This can be implemented with RSA signing. A traditional RSA signature is computed by raising the message m to the secret exponent d modulo the public modulus N.
See Blind RSA Signatures (https://en.wikipedia.org/wiki/Blind_signature#Blind_RSA_signatures)
but there are attacks against this scheme.

Confidential Transactions

Pedersen Commitments use additive homomorphic encryption is used to create confidential transactions.
Implementation by Elements (https://elementsproject.org/features/confidential-transactions/investigation)

Accumulators

- A cryptographic accumulator is a one way membership function. It consists of an algorithm to combine a large set of values into one, short accumulator, such that there is a short constant size witness that a given value was indeed incorporated into the accumulator.
  They can be used to prove membership and non membership of a set.

Dynamic accumulators allow dynamic updates to the set.

For an overview see Accumulator overview (https://cs.nyu.edu/~fazio/research/publications/accumulators.pdf) and Set membership (https://eprint.iacr.org/2015/404.pdf)

Threshold Cryptosystems / Secret Sharing / Multiparty Computation

The goal is to divide secret S into n pieces of data $S_i..S_n$ in such a way that:

Knowledge of any k or more S_i pieces makes S easy to compute. That is, the complete secret S can be reconstructed from any combination of k pieces of data.
Knowledge of any $k-1$ or fewer S_i pieces leaves S completely undetermined, in the sense that the possible values for S seem as likely as with knowledge of 0 pieces.

A naive splitting of a key would just make a brute force attack easier.

Shamir Secret Sharing

Based on the fact the k points are required to define a polynomial of degree $k-1$
With our points being elements in a finite field $\mathbb{F}$ of size P where $0 < klen < P; S < P$ and P is a prime number.
Choose at random $k-1$ positive integers $a_1..a_{k-1}$ with $a_i < P$ and let $a_0 = S$ Build the polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$$

Let us construct any n points out of it, for instance set $i = 1..n$ to retrieve $(i, f(i))$.
Every participant is given a point (a non-zero integer input to the polynomial, and the corresponding integer output) along with the prime which defines the finite field to use.
Given any subset of k of these pairs, we can find the coefficients of the polynomial using interpolation. The secret is the constant term a_0

Properties of Shamir's (k, n) threshold scheme are:

- Secure: Information theoretic security.
- Minimal: The size of each piece does not exceed the size of the original data.
- Extensible: When k is kept fixed, D_i pieces can be dynamically added or deleted without affecting the other pieces.
- Dynamic: Security can be easily enhanced without changing the secret, but by changing the polynomial occasionally (keeping the same free term) and constructing new shares to the participants.
- Flexible: In organizations where hierarchy is important, we can supply each participant different number of pieces according to their importance inside the organization. For

instance, the president can unlock the safe alone, whereas 3 secretaries are required together to unlock it.

Additive Secret Sharing

Given a secret $s \in F$, the dealer D selects $n-1$ random integers $R = r_1, r_2, r_{n-1}$ uniformly from F.

D then computes

$s_n = s - \sum_{i=1}^{n-1} r_i \ mod\ F$

D sends each player P_i 1 ≤ i ≤ n − 1 the share $s_i = r_i$, and the share s_n is sent to P_n.

The reconstruction of secret s ∈ F is trivial; simply add all of the shares together:

s = $\sum_{i=1}^{n} s_i \ mod\ F$

The above additive secret sharing scheme requires all participants to contribute their shares in order to reconstruct the secret.
If one or more of the participants are missing, no information about the original secret can be recovered; such a scheme is known as a perfect secret sharing scheme.

Multiparty computation

A key point to understand is that MPC is not a single protocol but rather a growing class of solutions that differ with respect to properties and performance. However, common for most MPC systems are the three basic roles:

- The Input Parties delivering sensitive data to the confidential computation.
- The Result Parties receiving results or partial results from the confidential computation.
- The Computing Parties jointly computing the confidential computation

In an multi party computation, a given number of participants, $p_1, p_2, \ldots p_n$, each have private data, respectively $d_1, d_2, \ldots d_n$.
Participants want to compute the value of a public function on that private data: $f(d_1, d_2 \ldots d_n)$ while keeping their own inputs secret.

Most MPC protocols make use of a secret sharing scheme such as Shamir Secret Sharing. The function $f(d_1, d_2 \ldots d_n)$ is defined as an “arithmetic circuit” over a finite field which consists of addition and multiplication gates.
In the secret sharing based methods, the parties do not play special roles. Instead, the data associated with each wire is shared amongst the parties, and a protocol is then used to evaluate each gate.

Secret sharing allows one to distribute a secret among a number of parties by distributing shares to each party. Two types of secret sharing schemes are commonly used;

1. Shamir secret sharing
2. Additive secret sharing

In both cases the shares are random elements of a finite field that add up to the secret in the field; intuitively, security is achieved because any non-qualifying set of shares looks randomly distributed.