PP=(EC **secp256k1**; BasePoint-Generator ***G***; prime ***p***; param. ***a***, ***b***); Parameters ***a***, ***b*** defines EC equation $y^2=x^3+ax+b \bmod p$ over F_p.
PrK $_{ECC}$=**x**; >> **x**=randi(***p***).
PuK$_{ECC}$=***A***=**x**∗***G***.
Alice ***A***: **x**=**….**; ***A***=(x_A, y_A);

H

$H(M) = h$

$M \in \{0,1\}^*$; $h \in \{0,1\}^{256}$

$|h| = 256$ bit

Let ***u***, ***v*** are integers < ***p***.

Property 1: $(u + v)*P = u*P \boxplus v*P$ replacement to --> $(u + v)P = uP + vP$

Property 2: $(u)*(P \boxplus Q) = u*P \boxplus u*Q$ replacement to --> $u(P + Q) = uP + uQ$

5.2 One-time addresses

Alice: Has ***Bob***'s public keys

PuK$_1$ = ***B***;

PuK$_2$ = ***D***;

Bob: **y** <-- randi(***p***); **PuK**$_1$ = ***B*** = **y**∗***G*** = (x_B, y_B);

z <-- randi(***p***); **PuK**$_2$ = ***D*** = **z**∗***G*** = (x_D, y_D);

Every Monero user has a pair of public keys as public address.
However, these addresses is never used directly.
Instead, new addresses based on a Diffie-Hellman-like exchange are created every time an amount is to be paid to a user.
In this way, external observers will not be able to identify receivers in transactions.
Imagine a very simple transaction, containing exactly one input and one output - a payment from ***Alice*** to ***Bob***.

Bob has private/public keys (**y**, **z**) and (***B***, ***D***).
To create one-time keys, ***Alice*** would proceed as follows:

1. ***Alice*** generates a random number ***r*** such that 1 < ***r*** < ***p***, and calculates the *output public key*

$$K_0 = H(r*B)*G + D.$$

h

2. ***Alice*** sets K_0 as the *addressee of the payment*, adds the value ***r***∗***G*** to the transaction data and submits it to the network.
The value ***r***∗***G*** will be used by the receiver to calculate a Diffiee-Hellman-like shared secret.

3. ***Bob*** receives the data and sees the value ***r***∗***G***. Hence, he can calculate

$$y*(r*G) = r*(y*G) = r*B.$$

Therefore, he will also be able to calculate

$$K_0 = H(r*B)*G + D.$$

When he sees the addressee of the output he will know that it is addressed to him.

4. The one-time keys for the output are

$$K_0 = H(r*B)*G + D = H(r*B)*G + z*G = [H(r*B) + z]*G.$$

4. The one-time keys for the output are

$$K_0 = H(r*B)*G + D = H(r*B)*G + z*G = [H(r*B) + z]*G.$$

$$k_0 = [H(r*B) + z].$$

While Alice can calculate the public key K_0 for the anonymous address, she cannot compute the corresponding private key k_0, since it would require either knowing Bob's second private key z, or solving the EC discrete logarithm problem for $D = z*G$, which we assume to be hard.

The private key y is often called the ***view key***.
The reason is that it allows a third party to verify if an output is addressed to ***Bob***, and yet, without the knowledge of the other private key z, this third party would not be able to spend the amount, as he would not be able to sign with the private key k_0 of the one-time address.
Such a third party could be a trusted custodian, an auditor, a tax authority, etc.
Somebody who would have read access to the user's transaction history, without any further rights.
This third party would also be able to decrypt the amounts using scheme like Pedersen commitment (of Section 5.4.1).
The private key k_0 can only be calculated with knowledge of z.
Spending an output will require calculating k_0, which in turn entails knowing the ***spend key*** z.

A: $r \leftarrow randi(p)$

$r*G$

$K_0 = H(r*B)*G + D$

$r*G$, K_0 →

B:

$y*(r*G) = r*(y*G) = r*B.$

$K_0 = H(r*B)*G + D = H(r*B)*G + z*G = [H(r*B) + z]*G.$

$k_0 = [H(r*B) + z].$

5.2.1 Multi-output transactions
Most transactions will contain more than one output.
If nothing else, to transfer back any change to the sender himself.
Monero senders generate only one random value r.
The session public parameter $r*G$ is normally known as the Transaction public key and is published in the blockchain.
To ensure that all output addresses in a transaction are different even in cases where the same addressee is used twice, Monero uses the output index.
Every output will have an index i in (1, 2, 3, …, n).
By appending this value to the shared secret before hashing it, one can ensure that the resulting addresses will be unique:

$$K_0 = H(r*B||i)*G + D = H(r*B||i)*G + z*G = [H(r*B||i)*G + z]*G.$$

$$k_0 = [H(r*B||i)*G + z].$$