5. Confidentiality and Anonymization-deanonymization in blockchain based on ECC

ECDSA-secp256k1.pdf
https://doi.org/10.1007/978-3-031-49099-6_22

In this section we present anonymization-deanonymization system based on Elliptic Curve Schnorr Signature Algoritm (EC SSA), (Jain and Pilli, 2023)

Jain, A., Pilli, E.S. (2023). SoK: Digital Signatures and Taproot Transactions in Bitcoin. In: Muthukkumarasamy, V., Sudarsan, S.D., Shyamasundar, R.K. (eds) Information Systems Security. ICISS 2023. Lecture Notes in Computer Science, vol 14424. Springer, Cham. https://doi.org/10.1007/978-3-031-49099-6_22

EC Schnorr Signature Algorithm (SSA) was added to Bitcoin in 2021. This scheme allows signatures aggregation and NIZKP.

We will follow the notations and definitions presented in Section 2 and 4 to better understand the similarities between Scnorr signature being constructed on the base of multiplicative group $\boldsymbol{Z_p^*}$. $\boldsymbol{F_p}$ EC Schnorr signature constructed on additive group of EC points defined in the finite field $\boldsymbol{F_p}$.
We are considering standardized EC denoted by secp256k1 and adopted in Bitcoin and other cryptocurrencies. Public Parameters (PP) consist of the following set

PP={EC secp256k1; BasePoint-Generator G; prime p; parameters a, b}, (5.1)
where prime p defines the finite field $\boldsymbol{F_p}$ ={0, 1, 2, ..., p-1} with characteristic of prime p;

Parameters a, b defines the equation of EC in $\boldsymbol{F_p}$

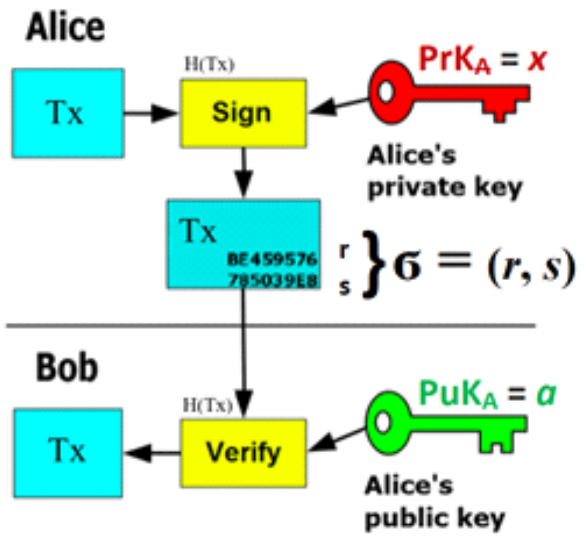
$$y^2=x^3+ax+b \bmod p. \quad (5.2)$$

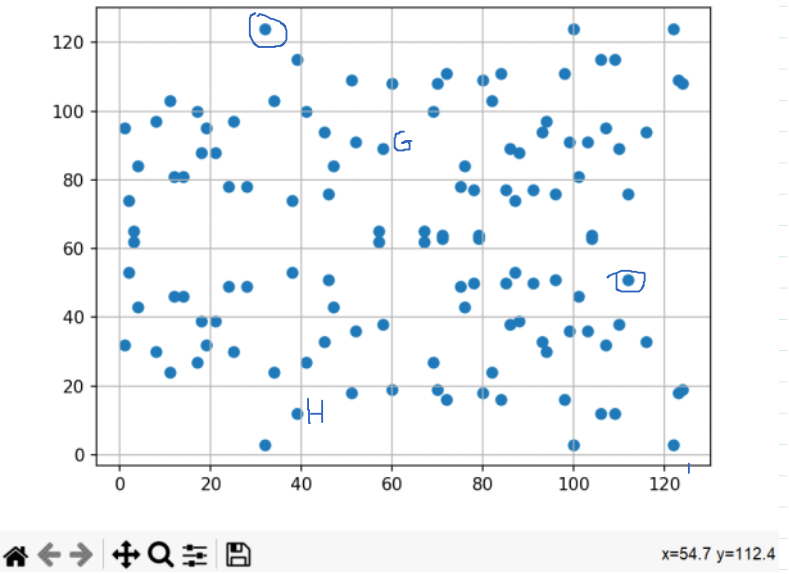
Operation	Comment
$\oplus$	Addition operation of 2 points in EC
$*$	Multiplication operation of scalar by EC point

Private key we denote by PrK and public key by PuK. PrK is generated at random and PuK is computed in the following way

$$\text{PrK} = z \leftarrow \text{rand}(\boldsymbol{F_p}), \quad (5.3)$$
$$\text{PuK} = A = z*G. \quad (5.4)$$

Notice that by A we denoted an EC point found by z times adding EC generator point G. We will follow the traditional notations when the integers in $\boldsymbol{F_p}$ are denoted by lowercase letters and EC points by capital letters. Temporary we will use non-traditional notations for operations between integers and EC points presented in the table.





Let a, b be any scalars in $\boldsymbol{F_p}$ and P is any point of EC. Then the following identity is used to create signatures in ECC:

$$(a+b)*P = a*P \oplus b*P. \qquad (5.5a)$$
$$(ab)*P = a*(b*P) = a*Q; \text{ if } Q = b*P. \qquad (5.5b)$$

Let M be a message to be signed and H-function SHA256 is used for signature creation on the h-value h of this function symbolically written in the following:

$$h = \text{SHA256('M')}. \qquad (5.6)$$

ECSSA is constructed using the following steps.

1. The random integer i in $\boldsymbol{F_p}$ is generated
$$i = \text{rand}\,(\boldsymbol{F_p}). \qquad (5.7)$$
2. Using generator G the EC point R is computed as a first component of signature
$$R = i*G = (x_R, y_R). \qquad (5.8)$$
3. The second component of signature is the scalar s computed by the equation
$$s = i + hz \mod p. \qquad (5.9)$$

Then the signature σ is expressed by the following two components

$$\sigma = (R, s) \qquad (5.10)$$

Symbolically the ECSSA signature we denote by the following notation

$$\text{Sign}_{EC}(z, h) = \sigma = (R, s). \qquad (5.11)$$

The verification of this signature is provided by the following relation

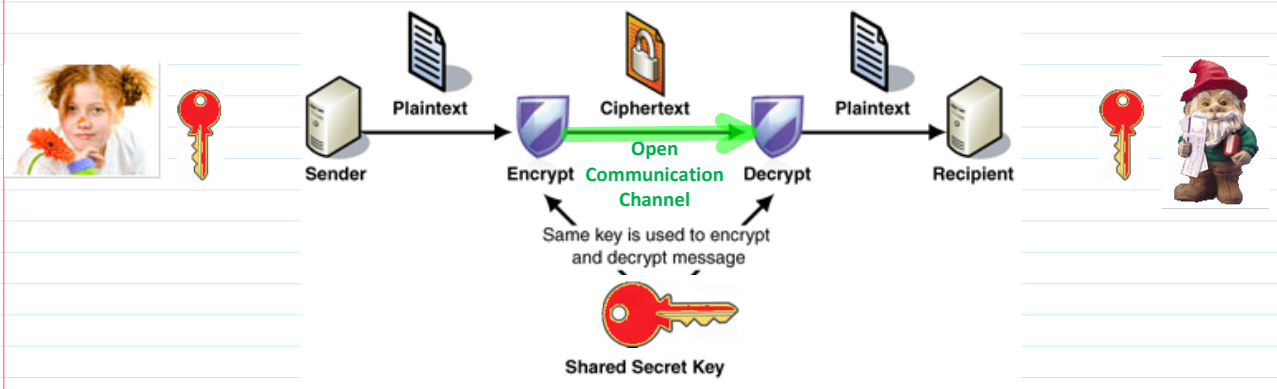
$$s*G = R \oplus h*A. \qquad (5.12)$$

Correctness follows from the identity (5.5a), (5.5b).

$$s*G = (i + hz)*G = i*G \oplus (hz)*A = R \oplus h*(z*G) = R \oplus h*A. \qquad (5.13)$$

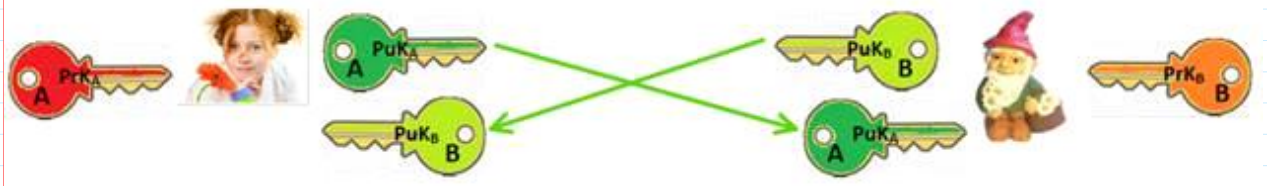
Analogously to classical Schnorr signature security of EC SSA relies on the difficulty of solution of problem to find z in (5.4) when A and G are given or to find i in (5.8) when R and G are given. This problem following this analogy is named

as EC discrete logarithm problem (DLP).



Alice: **PrK_A = z; PuK_A = z*G = A**
PuK_B = B

Bob: **PrK_B = y; PuK_B = y*G = B**
PuK_A = A



$u \leftarrow rand(F_p)$
$T_A = u * G$
$h_A = sha256('T_A')$
$Sign(z, h_A) = \sigma_A = (r_A, s_A)$

$T_A, \sigma_A = (r_A, s_A)$ →

$Ver(A, \sigma_A, h_A) = True$
$v \leftarrow rand(F_p)$
$T_B = v * G$
$h_B = sha256('T_B')$
$Sign(y, h_B) = \sigma_B = (r_B, s_B)$

← $T_B, \sigma_B = (r_B, s_B)$

$Ver(B, \sigma_B, T_B) = True$

$K_{AB} = u * T_B = u*(v*G) =$
$= (u\ v)*G$ $=$ K $=$ $K_{BA} = v * T_A = v*(u*G) =$
$= (v\ u) * G$

Let ***Alice*** computed the following T_A value as a point of EC with coordinates (x, y):
e0d473945a263cc22970731ba3070472358e514eff1f78464610ad07a952cece coordinate x
6c08280f3559a79996ad2839143e252ef7b90da5e284cc73cf3d8922741baf91 coordinate y

Then to sign this value T_A she computes h-value h_A.
>> hA=sha256('e0d473945a263cc22970731ba3070472358e514eff1f78464610ad07a952cece6c08280f35
59a79996ad2839143e252ef7b90da5e284cc73cf3d8922741baf91')

hA = 38CC536D27E3890984BD3737B91429701A8463D5A28E2592F4B8B3FCDE5D3E5F
>> length(hA)
ans = 64 % length of h-value is 64 hexadecimal digits, i.e. 256 bits corresponding to function sha256

The signature ***Alice*** is placing EC SSA on this h-value h_A.

SignCryption

1. Autenticated KAP: agreed symmetric secret key ***K***.
2. Encryption of finite length message ***M*** with symmetric cipher, e.g. AES128 and providing **confidentiality** of ***C***:
   $C = E(K, M)$
3. Hashing ciphertext ***C*** by obtaining h-value h_C of 256 bit length, e.g. sha256 providing **integrity**:
   $h_C = SHA256(C)$
4. Signing h_C with EC SSA using **PrK** = ***x*** providing **authenticity**:
   $\sigma = (R, s) = Sign_{EC}(x, h_C)$

Encrypt and **Sign** ***paradigm*** provides security against Chosen Ciphertext Attack - **CCA**:
It is most powerful attack but its realization is mostly complicated.

Pedersen commitments

Generally speaking, a cryptographic commitment scheme is a way of publishing a commitment to a value without revealing the value itself.
As an example, in a flip-coin game, Alice could commit to one outcome before Bob flips the coin, by publishing the value hashed with secret data.
After flipping the coin, Alice could prove which value she committed to by publishing her secret data, so that Bob could verify that she did indeed hash the outcome she later declared.
In other words, assume that Alice has a secret string ***s*** and that the value she wants to commit to is ***v***.
She could simply hash H(***s***||***v***) and tell the result to Bob.
Bob flips the coin and then Alice could prove that she guessed the right outcome ***v*** by telling Bob what the secret string ***s*** was.
Bob would then recalculate H(***s***||***v***) and verify that Alice did indeed guess right.

$(a+b)*P = a*P \oplus b*P$ ---> $C(a+b) = C(a) + C(b)$.

A *Pedersen commitment* [20] is a commitment that has the property of being *additive*. In other words, if $C(a)$ and $C(b)$ denote the commitments for amounts a and b respectively, then $C(a + b) = C(a) + C(b)$ is a commitment for $a + b$. This property is useful when committing transaction amounts, as one could prove, for instance, that inputs equal outputs, without unveiling the amounts at hand.

Fortunately, Pedersen commitments are easy to implement with elliptic curve cryptography, as the following holds trivially

$$aG + bG = (a + b)G$$

EC DLP: $aG = A \in EC$
it is infeasible to find a when G and A are given.

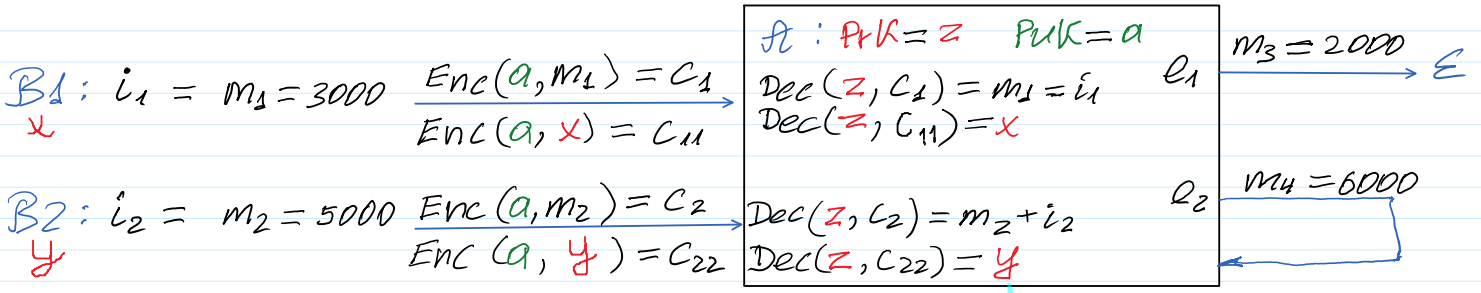
To attain information-theoretical privacy, one needs to add a secret *blinding factor* and another generator H, such that it is unknown for which value of γ the following holds $H = \gamma G$. The hardness of the discrete logarithm problem (D.L.P.) ensures the unfeasibility of calculating this value.

We can then define the commitment of an amount a as $C(x, a) = xG + aH$, where x is a blinding factor.

$$(a+b)\,c = ac + bc$$
$$(a+b)\,G = aG + bG$$

In the case of Monero, $H = to\_point(Keccak(G))$, where *Keccak* stands for the novel hashing algorithm of the same name, and *to_point* is a function mapping scalars to curve points.

UTXO – Unspent Transaction Output: $i_1 + i_2 = e_1 + e_2$ // honest transaction



B1: $x \leftarrow randi$

$C_{i1} = xG + i_1 H$

B2: $y \leftarrow randi$

$C_{i2} = yG + i_2 H$

C_{i1}, C_{i2} → Net ← C_{e1}, C_{e2}

A: $u \leftarrow randi$

$C_{e1} = uG + e_1 H$

$v \leftarrow randi$

$C_{e2} = vG + e_2 H$

Net: $C_{i1} + C_{i2} - (C_{e1} + C_{e2})$

$xG + i_1 H + yG + i_2 H - (uG + e_1 H + vG + e_2 H)$

$xG + i_1 H + yG + i_2 H - uG - e_1 H - vG - e_2 H$

$xG + yG - (uG + vG) + i_1 H + i_2 H - e_1 H - e_2 H$

$xG + yG - (uG + vG) + (i_1 + i_2) H - (e_1 + e_2) H$

$xG + yG - (uG + vG) + (i_1 + i_2 - (e_1 + e_2)) H$

$xG + yG - (uG + vG) + 0 \cdot H$; 0 – nulinis EC taškas

A

$C_{i12} = C_{i1} + C_{i2} =$

$= xG + i_1 H + yG + i_2 H$

$C_{e12} = C_{e1} + C_{e2} =$

$= uG + e_1 H + vG + e_2 H$

C_{i12}, C_{e12} → Net: verifies if $xG + yG - (uG + vG)$

$C_{i1} + C_{i2} - (C_{e1} + C_{e2}) = C_{i12} - C_{e12} =$

$= xG + yG - (uG + vG)$

$c_{i1} + c_{i2} - (c_{e1} + c_{e2}) = c_{i12} - c_{e12} =$

$= xG + yG - (uG + vG)$

$= uG + e_1 H + vG + e_2 H$