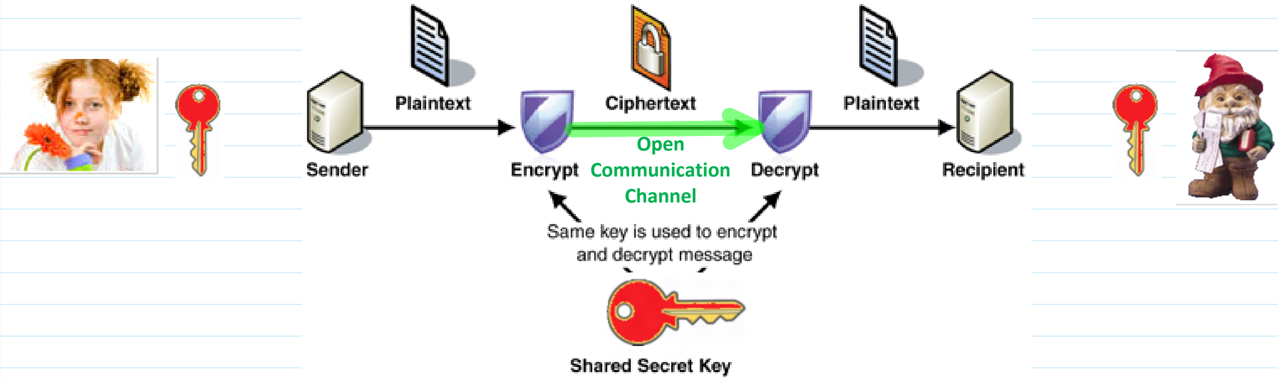

Alice: $PrK_A = z$; $PuK_A = z*G = A$
$PuK_B = B$

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

$u \leftarrow rand(\mathbb{Z}_p)$

$T_A = u * G$

$h_A = sha256('T_A')$

$Sign(z, T_A) = \sigma_A = (r_A, s_A)$

$$\xrightarrow{T_A,\ \sigma_A = (r_A, s_A)}$$

$Ver(A, \sigma_A, T_A) = True$

$v \leftarrow rand(\mathbb{Z}_p)$

$T_B = v * G$

$h_B = sha256('T_B')$

$Sign(y, h_B) = \sigma_B = (r_B, s_B)$

$$\xleftarrow{T_B,\ \sigma_B = (r_B, s_B)}$$

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

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

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 she computes h-value h_A.
>> hA=sha256('e0d473945a263cc22970731ba3070472358e514eff1f78464610ad07a952cece6c08280f3559a79996ad2839143e252ef7b90da5e284cc73cf3d8922741baf91')

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

The signature ***Alice*** is placing 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 *C***:

$\boldsymbol{C}$=E($\boldsymbol{k}$, $\boldsymbol{M}$)

3. Hashing ciphertext $\boldsymbol{C}$ by obtaining h-value $\boldsymbol{h_M}$ of 256 bit length, e.g. sha256 providing **integrity**:

$\boldsymbol{h_C}$ = SHA256($\boldsymbol{C}$)

4. Signing $\boldsymbol{h_M}$ with ECDSA using **PrK** = $\boldsymbol{x}$ providing **authenticity**:

$\boldsymbol{\sigma}$ = ($\boldsymbol{r}$, $\boldsymbol{s}$) = Sign($\boldsymbol{x}$, $\boldsymbol{h_M}$)

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 *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 (DLP) 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

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

$\mathcal{A}$: PrK $= z$ PuK $= a$

B1: $i_1 = m_1 = 3000$ $\xrightarrow{Enc(a, m_1) = c_1}$ $Dec(z, c_1) = m_1 = i_1$ e_1 $\xrightarrow{m_3 = 2000}$ $\mathcal{E}$

$Enc(a, x) = c_{11}$ $Dec(z, c_{22}) = x$

B2: $i_2 = m_2 = 5000$ $\xrightarrow{Enc(a, m_2) = c_2}$ $Dec(z, c_2) = m_2 + i_2$ e_2 $m_4 = 6000$

$Enc(a, m_2) = c_{22}$ $Dec(z, c_{22}) = y$

B1: $x \leftarrow randi$

$C_{i1} = xG + i_1 H$

B2: $y \leftarrow randi$

$C_{i2} = yG + i_2 H$

$$C_{i1}, C_{i2} \longrightarrow Net \longleftarrow C_{e1}, C_{e2}$$

$\mathcal{A}$: $u \leftarrow randi$

$C_{e1} = uG + e_1 H$

$v \leftarrow randi$

$C_{e1} = vG + e_2 H$

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

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

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

$$xG + yG - (uG + vG) + \underbrace{i_1 H + i_2 H}\, - \underbrace{e_1 H - e_2 H}$$

$$xG + yG - (uG + vG) + \underbrace{(i_1 + i_2) H - (e_1 + e_2) H}$$

$$\text{— '' —} \quad + (i_1 + i_2 - (e_1 + e_2)) H$$

$$\underbrace{xG + yG} - \underbrace{(uG + vG)} + 0 \cdot H = 0 \text{ – nulinis EC taškas}$$

$\mathcal{A}$

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

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

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

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

$$\xleftarrow{C_{i12},\ C_{e12}}$$

$\underbrace{xG + yG} - \underbrace{(uG + vG)}$

Net: verifies if

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

$$= \underbrace{xG + yG} - \underbrace{(uG + vG)}$$