On the base of ElGamal signature Elliptic Curve Digital Signature Algorithm - ECDSA is created.

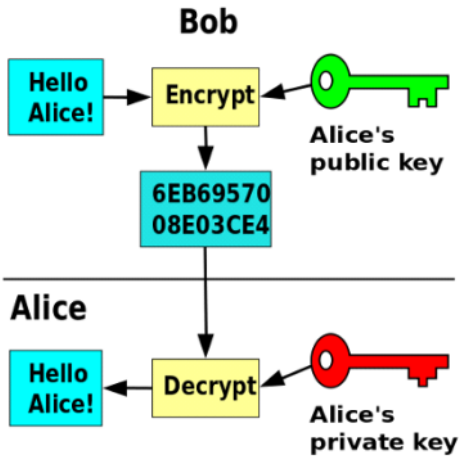
ECDSA is used in blockchain technologies.

Asymmetric Cryptography

Asymmetric Encryption - Decryption

c=Enc(PuK_A, m)

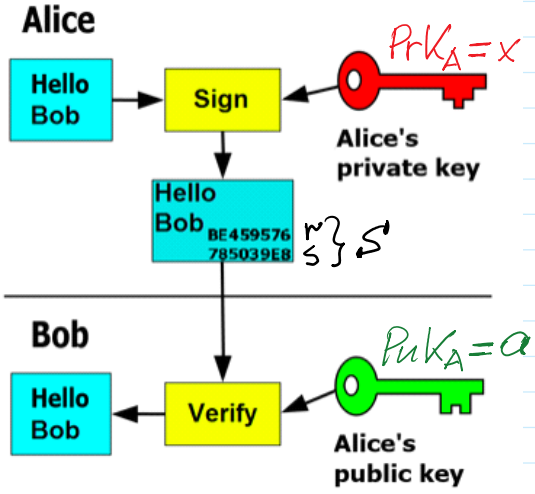
m=Dec(PrK_A, c)



Asymmetric Signing - Verification

S=Sig(PrK_A, m)

V=Ver(PuK_A, S, m), V∈{True, False}≡{1, 0}



El-Gamal E-Signature

1.*Public parameters generation*:

1.1.Strong prime number ***p*** generation. Prime number ***p*** is Strong prime if ***p***=2***q***+1, where ***q*** is prime.

1.2.Find a generator ***g*** in the group Z_p^*= {1, 2, 3, …, ***p***-1}, where multiplication operation is performed mod ***p***.

Number ***g*** is a generator of Z_p^*={1, 2, 3, …, ***p***-1} iff $g^q \neq 1 \bmod p$ and $g^2 \neq 1 \bmod p$.

1.3.Declare **Public Parameters** to the network **PP** = (***p***, ***g***); $p \sim 2^{2048} \rightarrow |p| = 2048$ bits

2.*Key generation*

- Randomly choose a private key ***x*** with **1 < *x* < *p*-2**.
- Compute $a = g^x \bmod p$.
- The private key is $PrK_A = x$.
- The public key is $PuK_A = a$.

3.***Signature creation***

To sign any finite message ***M*** the signer performs the following steps using public parametres **PP**.

- Compute **h=H(*M*)**.
- Choose a random ***k*** such that **1 < *k* < *p* - 2** and **gcd(*k*, *p* - 1) = 1**.
- Compute k^{-1} **mod (*p*-1): k^{-1} mod (*p*-1) exists** if **gcd(*k*, *p* - 1) = 1**, i.e. ***k*** and ***p*-1** are relatively prime.

k^{-1} can be found using either Extended Euclidean algorithmt or Euler theorem or .....

>> k_m1=mulinv(k,p-1) % k^{-1}mod (p-1) computation.

- Compute $r=g^k \bmod p$
- Compute $s=(h-xr)k^{-1} \bmod (p-1)$ --> $h=xr+sk \bmod (p-1)$.

Fermat's little theorem implies that all operations in the exponent are computed mod (*p*-1)

- Signature **S=Sig=(r,s)**

$$s=(h-xr)k^{-1} \bmod (p-1) \;/\!*k \Rightarrow sk=(h-xr)\cdot 1 \bmod (p-1) \;/\!+xr$$
$$\Rightarrow sk+xr=h \bmod (p-1) \Rightarrow h=sk+xr \bmod (p-1)$$

4.***Signature Verification***

A signature **S=Sig=(r,s)** on message ***M*** is verified using Public Parameters **PP**=(**p**, **g**) and **PuK$_A$**=**a**.

1. Bob computes h=H(M).
2. Bob verifies if $1<r<p-1$ and $1<s<p-1$.
3. Bob verifies if $V1=a^r r^s \bmod p$, $V2=g^h \bmod p$ and $V1=V2$.

The verifier Bob accepts a signature if all conditions are satisfied and rejects it otherwise.

5.**Correctness**

The algorithm is correct in the sense that a signature generated with the signing algorithm will always be accepted by the verifier.

The signature generation implies

$$h=xr+ks \bmod (p-1)$$

Fermat's little theorem implies that all operations in the exponent are computed mod (*p*-1)

$$g^h=g^{(xr+ks) \bmod (p-1)} \bmod p = g^{xr}g^{ks} = (\underbrace{g^x}_{a})^r(\underbrace{g^k}_{r})^s = a^r r^s \bmod p$$

El-Gamal E-Signature: using 24 bits arithmetics.

>> p=genstrongprime(24) p=int64(15728302)

```
>> p=genstrongprime(24)
p = 15728303
>> q=(p-1)/2
q =  7864151
>> isprime(p)
ans = 1
>> isprime(q)
ans = 1
>> pb=dec2bin(p)
pb = 1110 1111 1111 1110 1010 1111
>> ph=dec2hex(p)
ph = EFFEAF
```

p=int64(15728302)

g = 5

```
>> mod_exp(g,q,p)
ans = 15728302
>> mod_exp(g,2,p)
ans = 25
```

PP = (**p**=**15728303**, **g**=**5**);

$Z_{15}' = \{1, 2, 3, \ldots, 14\}$ Operations performed mod 15

Multiplication Tab.		Z15'												
*	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	2	4	6	8	10	12	14	1	3	5	7	9	11	13
3	3	6	9	12	0	3	6	9	12	0	3	6	9	12
4	4	8	12	1	5	9	13	2	6	10	14	3	7	11
5	5	10	0	5	10	0	5	10	0	5	10	0	5	10
6	6	12	3	9	0	6	12	3	9	0	6	12	3	9
7	7	14	6	13	5	12	4	11	3	10	2	9	1	8
8	8	1	9	2	10	3	11	4	12	5	13	6	14	7
9	9	3	12	6	0	9	3	12	6	0	9	3	12	6
10	10	5	0	10	5	0	10	5	0	10	5	0	10	5
11	11	7	3	14	10	6	2	13	9	5	1	12	8	4
12	12	9	6	3	0	12	9	6	3	0	12	9	6	3
13	13	11	9	7	5	3	1	14	12	10	8	6	4	2
14	14	13	12	11	10	9	8	7	6	5	4	3	2	1

```
>> k=8
k = 8
>> k_m1=mulinv(k,p-1)
k_m1 = Inverse element does not exist
>> gcd(k,p-1)
ans = 2
>> k=7
k = 7
>> k_m1=mulinv(k,p-1)
```

```
k_m1 = Inverse element does not exist
>> gcd(k,p-1)
ans = 2
>>
>> k=5
k = 5
>> k_m1=mulinv(k,p-1)
k_m1 = 3
>> mod(k*k_m1,p-1)
ans = 1
```