

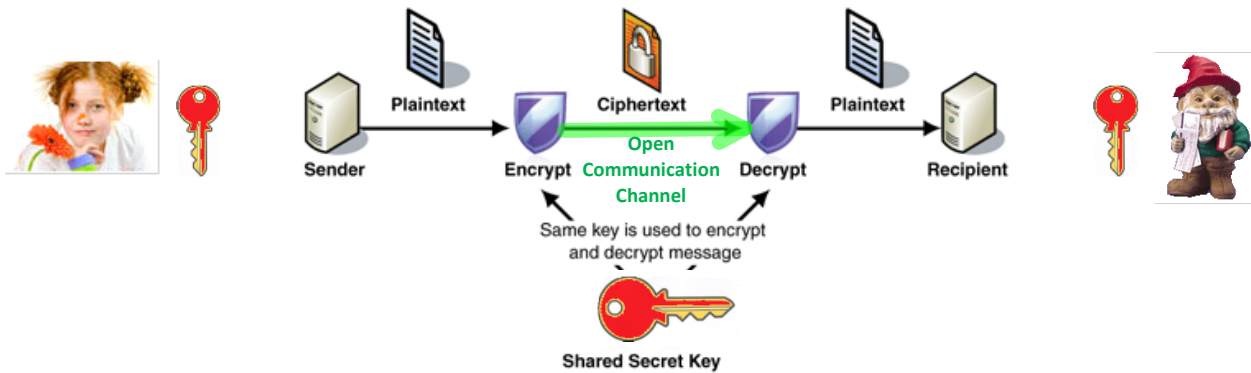
# Cryptography: Information confidentiality, integrity, authenticity, person identification

## Symmetric Cryptography ----- Asymmetric Cryptography Public Key Cryptography

Symmetric encryption  
H-functions, Message digest  
HMAC H-Message Authentication Code

Asymmetric encryption  
E-signature - Public Key Infrastructure - PKI  
E-money, Blockchain  
E-voting  
Digital Rights Management - DRM (Marlin)  
Etc.

### Symmetric - Secret Key Encryption - Decryption



### Public Key Cryptography - PKC

#### Principles of Public Key Cryptography

Instead of using single symmetric key shared in advance by the parties for realization of symmetric cryptography, asymmetric cryptography uses two *mathematically* related keys named as private key and public key we denote by **PrK** and **PuK** respectively.

**PrK** is a secret key owned *personally* by every user of cryptosystem and must be kept secretly. Due to the great importance of **PrK** secrecy for information security we labeled it in red color. **PuK** is a non-secret *personal* key and it is known for every user of cryptosystem and therefore we labeled it by green color. The loss of **PrK** causes a dramatic consequences comparable with those as losing password or pin code. This means that cryptographic identity of the user is lost. Then, for example, if user has no copy of **PrK** he get no access to his bank account. Moreover his cryptocurrencies are lost forever. If **PrK** is got into the wrong hands, e.g. into adversary hands, then it reveals a way to impersonate the user. Since user's **PuK** is known for everybody then adversary knows his key pair (**PrK**, **PuK**) and can forge his Digital Signature, decrypt messages, get access to the data available to

the user (bank account or cryptocurrency account) and etc.

Let function relating key pair (PrK, PuK) be  $F$ . Then in most cases of our study (if not declared opposite) this relation is expressed in the following way:

$$\text{PuK} = F(\text{PrK}).$$

In open cryptography according to Kerchoff principle function  $F$  must be known to all users of cryptosystem while security is achieved by secrecy of cryptographic keys. To be more precise to compute PuK using function  $F$  it must be defined using some parameters named as public parameters we denote by PP and color in blue that should be defined at the first step of cryptosystem creation. Since we will start from the cryptosystems based on discrete exponent function then these public parameters are

$$\text{PP} = (p, g).$$

Notice that relation represents very important cause and consequence relation we name as the direct relation: when given PrK we compute PuK.

Let us imagine that for given  $F$  we can find the inverse relation to compute PrK when PuK is given. Abstractly this relation can be represented by the inverse function  $F^{-1}$ . Then

$$\text{PrK} = F^{-1}(\text{PuK}).$$

In this case the secrecy of PrK is lost with all negative consequences above. To avoid these undesirable consequences function  $F$  must be **one-way function** – OWF. In this case informally OWF is defined in the following way:

1. The computation of its direct value PuK when PrK and  $F$  are given is effective.
2. The computation of its inverse value PrK when PuK and  $F$  are given is infeasible, meaning that to find  $F^{-1}$  is infeasible.

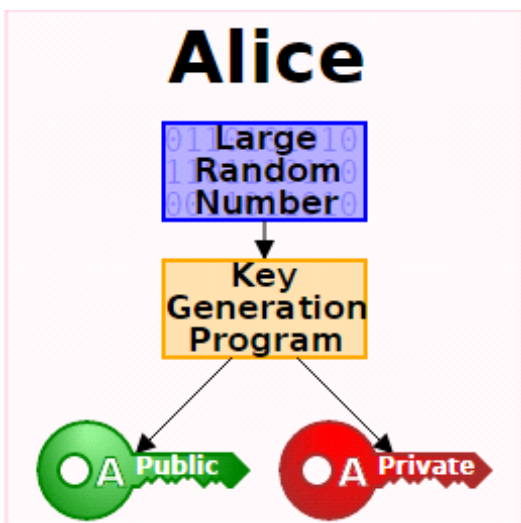
The one-wayness of  $F$  allow us to relate person with his/her PrK through the PuK. If  $F$  is 1-to-1, then the pair (PrK, PuK) is unique. So PrK could be reckoned as a unique secret parameter associated with certain person. This person can declare the possession or PrK by sharing his/her PuK as his public parameter related with PrK and and at the same time not revealing PrK.

So, every user in asymmetric cryptography possesses key pair (PrK, PuK). Therefore, cryptosystems based on asymmetric cryptography are named as **Public Key CryptoSystems** (PKCS).

We will consider the same two traditional (canonical) actors in our study, namely Alice and Bob.

Everybody is having the corresponding key pair (PrK<sub>A</sub>, PuK<sub>A</sub>) and (PrK<sub>B</sub>, PuK<sub>B</sub>) and are exchanging with their public keys using open communication channel as indicated in figure below.

## Asymmetric - Public Key Cryptography



PrK and PuK are related

$$\text{PuK} = F(\text{PrK})$$

$F$  is one-way function

Having PuK it is infeasible to find

$$\text{PrK} = F^{-1}(\text{PuK})$$

$F(x)=a$  is OWF, if:

1. It is easy to compute  $a$ , when  $F$  and  $x$  are given.
2. It is infeasible to compute  $x$  when  $F$  and  $a$  are given.

$$\text{PrK} = x \leftarrow \text{randi} \implies \text{PuK} = a = g^x \text{ mod } p$$

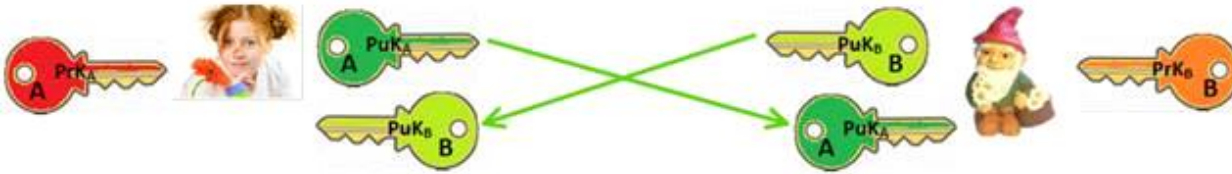
Public Parameters PP = (p, g)

$$p \sim 2^{2048} \implies |p| \cong 2048 \text{ bits}$$

$$p \sim 2^{28} \implies |p| \cong 28 \text{ bits}$$

Threats of insecure PrK generation

$$\mathcal{I}_p^* = \{1, 2, 3, \dots, p-1\}; * \text{ mod } p$$



Message  $m < p$

### Asymmetric Signing - Verification

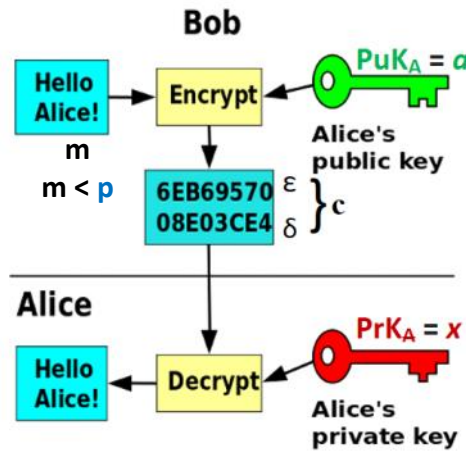
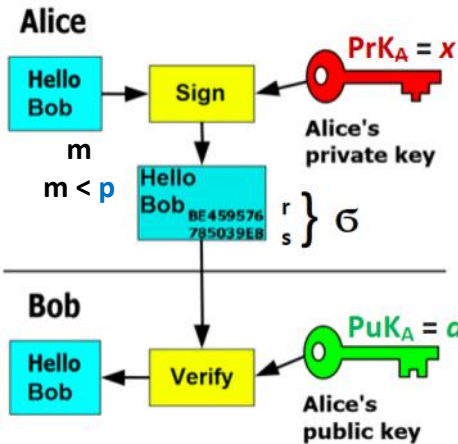
$$\text{Sign}(\text{PrK}_A, m) = \sigma = (r, s)$$

$$V = \text{Ver}(\text{PuK}_A, m, \sigma), V \in \{\text{True}, \text{False}\} \equiv \{1, 0\}$$

### Asymmetric Encryption - Decryption

$$c = \text{Enc}(\text{PuK}_A, m)$$

$$m = \text{Dec}(\text{PrK}_A, c)$$



## ElGamal Cryptosystem

### 1. Public Parameters generation $PP = (p, g)$ .

Generate strong prime number  $p$ : `>> p=genstrongprime(28)` % strong prime of 28 bit length

Find a generator  $g$  in  $Z_p^* = \{1, 2, 3, \dots, p-1\}$  using condition.

Strong prime  $p=2q+1$ , where  $q$  is prime, then  $g$  is a generator of  $Z_p^*$  iff

$$g^q \not\equiv 1 \pmod{p} \text{ and } g^2 \not\equiv 1 \pmod{p}$$

Declare Public Parameters to the network  $PP = (p, g)$ ;

$$p = 268435019; g = 2;$$

$$2^{28}-1 = 268,435,455$$

$$\gg 2^{28}-1$$

$$\text{ans} = 2.6844e+08$$

$$\gg \text{int64}(2^{28}-1)$$

$$\text{ans} = 268435455$$

$$\text{PrK} = x \leftarrow \text{randi} \implies \text{PuK} = a = g^x \pmod{p}$$

## El-Gamal E-Signature

The ElGamal signature scheme is a [digital signature](#) scheme which is based on the difficulty of computing [discrete logarithms](#).

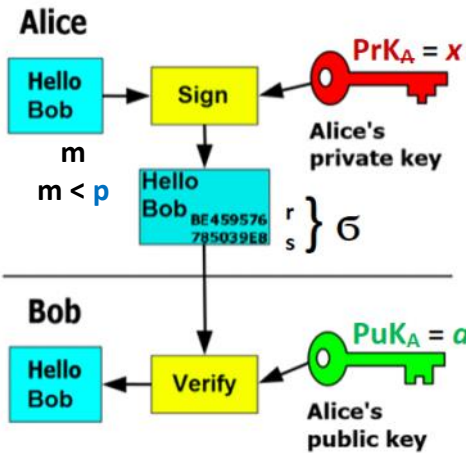
It was described by [Taher ElGamal](#) in 1984. The ElGamal signature algorithm is rarely used in practice.

A variant developed at [NSA](#) and known as the [Digital Signature Algorithm](#) is much more widely used.

The ElGamal signature scheme allows a third-party to confirm the authenticity of a message sent over an insecure channel.

From [https://en.wikipedia.org/wiki/ElGamal\\_signature\\_scheme](https://en.wikipedia.org/wiki/ElGamal_signature_scheme)

EC Gamal sign. → Digital Signature Alg. (DSA) NSA  
 → Elliptic Curve DSA - ECDSA Certicom



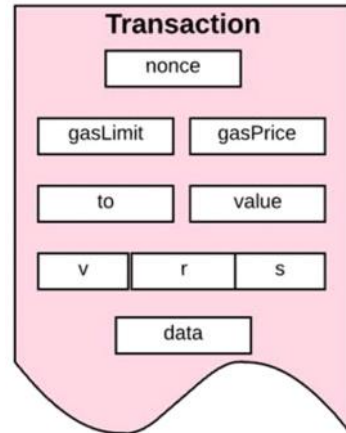
Signature creation for message  $M \gg p$ .

1. Compute decimal h-value  $h=H(M)$ ;  $h < p$ .
  2. Generate  $i = \text{int64}(\text{randi}(p-1))$  such that  $\text{gcd}(i, p-1) = 1$ .
  3. Compute  $i^{-1} \bmod (p-1)$ . You can use the function  $\gg i\_m1 = \text{mulinv}(i, p-1)$ ;
  4. Compute  $r = g^i \bmod p$ .
  5. Compute  $s = (h - xr)i^{-1} \bmod (p-1)$ .
  6. Signature on h-value  $h$  is  $\sigma = (r, s)$ .
- Sign(x, h) = sigma = (r, s).**

```
>> p=int64(genstrongprime(28))
```

```
>> p= int64(268435019)
p = 268435019
>> g=2
g = 2
```

```
>> i=randi(p-1)
i = 1.1728e+08
>> i=int64(randi(p-1))
i = 47250243
>> gcd(i,p-1)
ans = 1
>> i_m1=mulinv(i,p-1)
i_m1 = 172715821
>> mod(i*i_m1,p-1)
ans = 1
```



$T_x = \text{'nonce || gasLimit || gasPrice || to || value || data'}$   
 $h = H(T_x) \rightarrow \sigma = (r, s) = \text{Sign}(PrK, h)$

**1. 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  $i$  such that  $1 < i < p - 1$  and  $\text{gcd}(i, p - 1) = 1$ .
- Compute  $i^{-1} \bmod (p-1)$ :  $i^{-1} \bmod (p-1)$  exists if  $\text{gcd}(i, p - 1) = 1$ , i.e.  $i$  and  $p-1$  are relatively prime.  
 $k^{-1}$  can be found using either Extended Euclidean algorithm or Euler theorem or ....  
 $\gg i\_m1 = \text{mulinv}(i, p-1)$  %  $i^{-1} \bmod (p-1)$  computation.

• Compute  $r = g^i \bmod p$

• Compute  $s = (h - xr)i^{-1} \bmod (p-1) \rightarrow h = xr + is \bmod (p-1)$

Signature  $\sigma = (r, s)$

$s = (h - xr) \cdot i^{-1} \pmod p$   
 $s \cdot i = (h - xr) \cdot i^{-1} \cdot i \pmod p$   
 $h - xr = s \cdot i \rightarrow h = xr + i \cdot s \pmod p$

## 2. Signature Verification

A signature  $\sigma = (r, s)$  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 calculates  $V1 = g^h \bmod p$  and  $V2 = a^r r^s \bmod p$ , and verifies if  $V1 = V2$ .
- The verifier Bob **accepts** a signature if all **conditions** are satisfied during the signature creation and **rejects** it otherwise.

## 3. 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

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

Hence [Fermat's little theorem](#) implies that all operations in the exponent are computed mod  $(p-1)$

$$\begin{matrix} g^h \bmod p & = & g^{(xr+is) \bmod (p-1)} \bmod p & = & g^{xr} g^{is} & = & (g^x)^r (g^i)^s & = & a^r r^s \bmod p \\ \mathbf{V1} & & & & & & \mathbf{(a) (r)} & & \mathbf{V2} \end{matrix}$$

$PP = (p, g)$

Lo:  $z \leftarrow \text{randi}(p-1)$   
 $v = g^z \bmod p$

Dear B I am A  
 and I am sending  
 you my  $PuK = v$

B: believes that  
 $PuK = v$  is of A

$m = \text{'Bob get out'}$   
 $\sigma = \text{Sign}(z, m) = (r, s)$

$m, \sigma = (r, s) \rightarrow$  B: verifies the signature  $\sigma$   
 on  $m$  using  $PuK = v$  and  
 verification passes.

Before Bob verifies any signature with someone  $PuK$  he must be sure that this  $PuK$  is got from the certain person, e.g. A but not from anybody else!

It is achieved by creation of PKI - Public Key Infrastructure when Trusted Third Party (TTP) such as Certification Authority is introduced. CA is issuing  $PuK$  Certificates for any user by signing  $PuK$  when user proves his/her identity to CA.

A: Identification Card - ID  
 $PrK_A = x; PuK_A = a$

$PuK$  | cert



CA:  $PrK_{CA}; PuK_{CA}$

$\xrightarrow[\text{ID}]{PuK_A}$   $\text{Sign}(PrK_{CA}, PuK_A) = \sigma_A$

$PrK_A \rightarrow A; PubK_A \rightarrow U.$



Is sure that  $PubK_A$  is of  $A$

Since CA is TTP &  $B$  can download  $PubK_{CA}$  using his browser with known to everyone link

<https://certificationauthority.trusted.com>

<https://certicom.com>

More about that is in module P170 M100.

Till this place

```
>> p= int64(268435019)
p = 268435019
>> g=2;
>> x =int64(randi(p-1))
x = 65770603
>> a=mod_exp(g,x,p)
a = 232311991
>> M='Hello Bob...'
M = Hello Bob...
>> h=hd28(M)
h = 150954921
```

```
>> i =int64(randi(p-1))
i = 201156232
>> gcd(i,p-1)
ans = 2
>> i =int64(randi(p-1))
i = 35395315
>> gcd(i,p-1)
ans = 1
>> i_m1=mulinv(i,p-1)
i_m1 = 192754179
>> mod(i*i_m1,p-1)
ans = 1
```

```
>> r=mod_exp(g,i,p)
r = 172536234
>> hmxr=mod(h-x*r,p-1)
hmxr = 20262153
>> s=mod(hmxr*i_m1,p-1)
s = 44575091
```

```
>> g_h=mod_exp(g,h,p)
g_h = 241198023
>> V1=g_h
V1 = 241198023
>> a_r=mod_exp(a,r,p)
a_r = 49998673
>> r_s=mod_exp(r,s,p)
r_s = 111993804
>> V2=mod(a_r*r_s,p)
V2 = 241198023
```