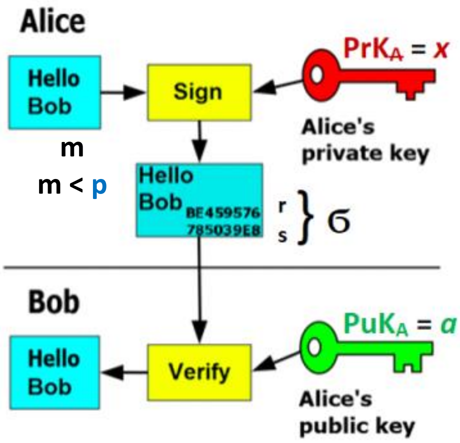
El-Gamal E-Sign

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>

EC Gamal sign. → Digital Signature Alg. (DSA) NSA
→ Elliptic Curve DSA – ECDSA Certicom: Menezes Vanstone

Declare **Public Parameters** to the network **PP** = (**p**, **g**); **p= 268435019**; **g=2;**

Real |p| = 2048 bits; Modeled |p| = 28 bits 2^28-1= 268,435,455



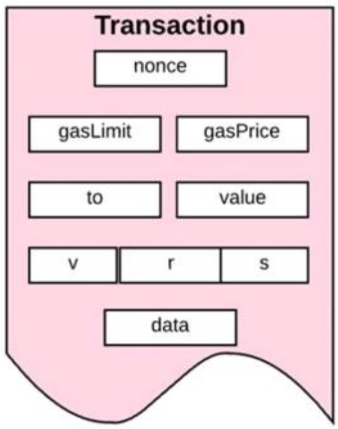
Signature creation for message M >> p.

1. Compute decimal h-value $h=H(M)$; $h<p$.
2. Generate >> i =int64(randi(p-1)) % such that gcd(i,p-1)=1.
3. Compute i^{-1} mod (p-1). You can use the function
   >> i_m1=mulinv(i, p-1);
4. Compute $r=g^i$ **mod** p.
5. Compute $s=(h-xr)i^{-1}$ mod (p-1).
6. Signature on h-value h is σ = (r,s)
   Sign(x,h) = σ = (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 = ' nonce || gasLimit || gasPrice || to || value || data '
$h = H(T_x) \longrightarrow \sigma = (r, s) = Sign(PrK, h)$

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 **gcd**(i, $p-1$) = 1.

- Compute i^{-1} **mod (p-1)**: i^{-1} **mod (p-1) exists** if gcd$(i, p-1) = 1$, i.e. i and **p-1** are relatively prime.

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

>> i_m1=mulinv(i,p-1) % i^{-1}mod (p-1) computation.

- Compute **$r = g^i \bmod p$**
- Compute **$s = (h - xr)i^{-1} \bmod (p-1)$ --> $h = xr + is \bmod (p-1)$**

Signature **$\sigma = (r,s)$**

$$\left.\begin{aligned} s &= (h - x\cdot r)\cdot i^{-1} \quad / \, i \\ s\cdot i &= (h - x\cdot r)\cdot \cancel{i^{-1}}\cdot\cancel{i} \\ h - xr &= s\cdot i \quad \Longrightarrow \quad h = x\cdot r + i\cdot s \end{aligned}\right\} \bmod p-1$$

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)

$$\underbrace{g^h \bmod p}_{V1} = g^{(xr+is) \bmod (p-1)} \bmod p = g^{xr} g^{is} = \underbrace{(g^x)}_{(a)}{}^{r}\underbrace{(g^i)}_{(r)}{}^{s} = \underbrace{a^r r^s \bmod p}_{V2}$$

Security PrK compromization: for given a, p, g find $PrK = x$ from the equation $a = g^x \bmod p$ $\quad / \, dlog_g$

$$dlog_g a = dlog_g (g^x \bmod p)$$

$$x\cdot(dlog_g g \bmod p) = dlog_g a$$

$$x\cdot 1 = dlog_g a$$

$$\boxed{x = dlog_g a}$$

Discrete Logarithm Problem (DLP)

1. Criteria: parameters (p, g) must be chosen in such a way that DLP must be infeasible.

But there exist such groups where DLP is feasible.

2. Let we have two random generated values $u, v \leftarrow rand(set)$

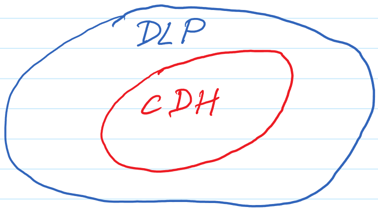
Compute value $g^{uv} = e$.

Let we chose $z \leftarrow rand(set)$ and compute $g^z = d$.

$(d, e) \longrightarrow$ verifier $\Longrightarrow$ it is infeasible to define

Let we chose $z \leftarrow rand(Set)$ and compute $g^z = d$.
$(d, e) \longrightarrow$ verifier $\Longrightarrow$ it is ifeasible to define either $d = g^z$ or $d = g^{uv}$.

computational Diffie-Hellman Assumption: CDH Assumption



CDH $\Longrightarrow$ DLP

CDH $\not\Longleftarrow$ DLP

$PP = (p, g)$

Ao: $z \leftarrow randi(p-1)$
$V = g^z \bmod p$

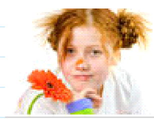
Dear B I am A and I am sending you my $PuK = V$ $\longrightarrow$ B: Believes that $PuK = V$ is of A

$m =$ 'Bob get out'
$\sigma = Sign(z, m) = (r, s)$ $\xrightarrow{m,\ \sigma = (r, s)}$ B: verifies the signature σ 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 Ifrastructure 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
$PrK_A = x$; $PuK_A = a$.



CA: PrK_{CA}; PuK_{CA}.

$\xrightarrow[PuK_A]{ID}$

$Sign(PrK_{CA}, PuK_A) = \sigma_A$
$Cert_A =$ 'σ_A, $Data_A$'

$\xleftarrow{cert_A}$

$PuK_A \downarrow Cert_A$

B: $Ver(PuK_{CA}, PuK_A, \sigma_A) = True$
Is sure that PuK_A is of A

Since CA is TTP & B can download PuK_{CA} using his browser with known to everyone link
https://CertificationAuthority.Tusted.com
https://certicom.com

```
>> p= int64(268435019)    >> i =int64(randi(p-1))    >> r=mod_exp(g,i,p)    >> g_h=mod_exp(g,h,p)
```

```
>> 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
```

imimsociety.net